APPENDIX C
COUNTERACTING GEOMETRIC DISTORTIONS IN WATERMARKING

ALIGN.CPP

```
/*******************************************************************\
* FILE: Align.cpp
*
* DESCRIPTION:
* Main source file for the Align class.  The Align class provides
* services related to aligning (synonomous with registering) a suspect
* image with a reference image.  The suspect requires some combination
* of translation, scaling, and rotation to achieve this.
*
* This version incorporates the Version 1.0 Alignment core algorithms
* from Geoff Rhoads, 2/17/96.
*
* Copyright (C) Digimarc Corporation, 1996, all rights reserved.
\*******************************************************************/
#include <math.h>

#include <memory.h>       // added by cld...
#include "stdafx.h"
#include "align.h"
#include "fft.h"


//////////////////////////////////////////////////////////////////////
// Align()
//
// Constructor for Align objects.
//////////////////////////////////////////////////////////////////////
Align::Align()
{
  m_alignStatus.x_scale = (float) 0.0;
  m_alignStatus.y_scale = (float) 0.0;
  m_alignStatus.x_trans = (float) 0.0;
  m_alignStatus.y_trans = (float) 0.0;
  m_alignStatus.rotation = (float) 0.0;
  m_alignStatus.refinement = (float) 0.0;
}

//////////////////////////////////////////////////////////////////////
// CORE ALGORITHMS FOLLOW
// The remainder of this file is devoted to the Align (i.e., register)
// core algorithms from Geoff Rhoads, modified slightly to comply with
// C++ and/or Windows programming standards.
//////////////////////////////////////////////////////////////////////

//#include <stdio.h>
// #include <stdlib.h>


#define START_RADIUS 0.10   /* ratio of nyquist at which log scale vectors are started */
#define PICK_RADIUS  7   /* radius of samples to ignore around previously found candidates */
#define START_RADIUS_1D 0.07   /* ratio of nyquist at which log scale vectors are started */
#define MAX_CANDIDATES  20  // this number can be set to 10 or even 50 when we start pushing things???
#define PI 3.141592653589
#define WINDOW_ORIGINALS 1
#define WINDOW_LOGPOLAR_LOG 1
#define MAX_LINEAR_DIMENSION 4096
#define SMALL (float)1e-10
#define REFINED_ROTATION_DIMENSION 512
#define REFINED_ROTATION_BITS 9
#define LOG_MOV_AVG 27
#define LOG_SMOOTH 3
#define NOMINAL_DOWNSAMPLE_DIM 256
#define SUPER_DOWNSAMPLE_DIM 128

#define SIGNATURE_BLOCK_DIMENSION 128
#define MELLIN_DIMENSION 128

int lp_sampling = 128;     /* total number of log-scale samples, should be plenty */
int lp_bits = 7;     /* bit value of above line */
double scale_increment;

float wr[MAX_LINEAR_DIMENSION],wi[MAX_LINEAR_DIMENSION];

extern int realfft2d_in_place(float *ar,int nbits,int inv,float *wr,float *wi );
extern void fft(float *ar,float *ai,int nbits,int inv,float *wr,float *wi,int neww);
extern int load_bump_array(
  float *bump,
  unsigned char *data,
  long xdim,
  long zdim,
  long bump_size,
  long jump_x,
  long overfill
);

int shift_array(float *array,int dim){
    int i,j;
    int dim2 = dim/2;
    int offset = dim2*dim + dim2;
    float *p1,*p2,ftmp;

    for(i=0;i<dim2;i++){
        p1 = &array[i*dim];
        p2 = &array[offset+i*dim];
        for(j=0;j<dim2;j++){
            ftmp = *p1;
            *p1 = *p2;
            *p2 = ftmp;
            p1++;p2++;
        }
    }
    offset = dim2*dim;
    for(i=0;i<dim2;i++){
        p1 = &array[dim2+i*dim];
        p2 = &array[offset+i*dim];
        for(j=0;j<dim2;j++){
            ftmp = *p1;
            *p1 = *p2;
            *p2 = ftmp;
            p1++;p2++;
        }
    }
    return(0);
}


int convert_to_magnitude(
    float *out,
    float *in,
    int dim
){
    int i,j,dim2 = dim/2;
    float *preal,*pimag,*pout,ftmp;

    preal = in;
    pimag = &in[dim];
    pout = out;
    for(i=0;i<(1+dim2);i++){
        for(j=0;j<dim;j++){
            ftmp = *preal * *preal + *pimag * *pimag;
            *pout = (float)sqrt( (double)ftmp );
            preal++;pimag++;pout++;
        }
        preal+=dim;
        pimag+=dim;
    }

return(1);
}


int convert_to_magnitude_1d_inplace(
    float *real,
    float *imaginary,
    int dim
){
    int i,dim2 = dim/2;
    float *preal,*pimag,ftemp;

    preal = real;
    pimag = imaginary;
    for(i=0;i<dim;i++){
        ftemp = *preal * *preal + *pimag * *pimag;
        *(preal++) = (float)sqrt( (double)ftemp );
        pimag++;
    }

return(1);
}


int log_polar_remap(
    float *in,
    float *out,
    int dim
){
    int i,dim2 = dim/2,xx,yy,j,jj,k;
    float *pin,*pout,ftemp[MAX_LINEAR_DIMENSION];
    double theta,dx,dy,radius[MAX_LINEAR_DIMENSION],x,y,fracx,fracy,*pradius;
```

```
    scale_increment=pow( 1.0/(double)START_RADIUS, 1.0/(double)lp_sampling);
    for(i=0;i<lp_sampling;i++){
        radius[i] = (START_RADIUS*(double)dim2) * pow(scale_increment,(double)i);
    }

    pout = out;
    for(theta=0.0,j=0;j<lp_sampling; j++,theta += (PI/lp_sampling)){
        dx = cos(theta);
        dy = sin(theta);
        pradius = radius;
        pout = &out[j];
        for(i=0;i<lp_sampling;i++){
            x = (double)dim2 + *pradius * dx;
            y = *(pradius++) * dy;
            xx = (int)x;
            yy = (int)y;
            fracx = x - (double)xx ;
            fracy = y - (double)yy ;
            pin = &in(yy*dim + xx);
            *pout = (float) ( (1.0-fracx)*(1.0-fracy)* (double)*(pin++) );
            *pout += (float) ( fracx*(1.0-fracy)* (double)*pin );
            pin += (dim-1);
            *pout += (float) ( (1.0-fracx)*fracy* (double)*(pin++) );
            *pout += (float) ( fracx*fracy * (double)*pin );
            pout += lp_sampling;
        }
    }

    /* now filter it along the scale axis */
    /* this generally increases the peak to noise ratio in finding the proper scale rotation */
    for(i=0;i<lp_sampling;i++){
        pout = ftemp;
        for(j=0;j<lp_sampling;j++){
            *pout = (float)0.0;
            for(k=-(LOG_MOV_AVG/2);k<=(LOG_MOV_AVG/2);k++){
                jj=j+k;
                if(jj<0)jj=0;
                else if(jj>= lp_sampling)jj=lp_sampling-1;
                *pout += out[i+jj*lp_sampling];
            }
            *(pout++)/=(float)LOG_MOV_AVG;
        }
        pin = ftemp;
        pout = &out[i];
        for(j=0;j<lp_sampling;j++){
            *pout -= *(pin++);
            pout += lp_sampling;
        }
        pout = ftemp;
        for(j=0;j<lp_sampling;j++){
            *pout = (float)0.0;
            for(k=-(LOG_SMOOTH/2);k<=(LOG_SMOOTH/2);k++){
                jj=j+k;
                if(jj<0)jj=0;
                else if(jj>= lp_sampling)jj=lp_sampling-1;
                *pout += out[i+jj*lp_sampling];
            }
            *(pout++)/=(float)LOG_SMOOTH;
        }
        memcpy(&out[i],ftemp,lp_sampling*sizeof(float) );
    }

return(1);
}



float get_median_float(float *median){
      if( median[0] > median[2] )return( -(median[0] - median[2])/(median[1] + median[0] -
2*median[2]) );
      else return( (median[2] - median[0])/(median[1] + median[2] - 2*median[0]) );
}



float get_2D_median(float *array,int xdim,int ydim,int high_x,int high_y,
                    float *x_offset,float *y_offset){
  float xmedian[3],ymedian[3],*py,*px,ratio,value;
  int j,jtemp,k,ktemp;

  ymedian[0]=ymedian[1]=ymedian[2]=(float)0.0;
  xmedian[0]=xmedian[1]=xmedian[2]=(float)0.0;
  py = ymedian;
  for(j=-1;j<2;j++){
      jtemp = high_y+j;
      if(jtemp < 0)jtemp=ydim-1;
      else if(jtemp==ydim)jtemp=0;
      px = xmedian;
      for(k=-1;k<2;k++){
          ktemp = high_x+k;
              if(ktemp < 0)ktemp=xdim-1;
              else if(ktemp==xdim)ktemp=0;
              *py += array[jtemp*xdim+ktemp];
              *(px+i) += array[jtemp*xdim+ktemp];
          }
          py++;
      }
      /* now find median values */
      ratio =  get_median_float(ymedian);
      *y_offset = (float)high_y + ratio;
      ratio =  get_median_float(xmedian);
      *x_offset = (float)high_x + ratio;
      value = (xmedian[0]+xmedian[1]+xmedian[2])/(float)9.0;

      return(value);
}


/* this is the fft window profile for mitigating edge effects; change to other windows if
their better */
/* or..., maybe certain windows are better for certain tasks, e.g., log polar vs. straight
correlation */
int load_windowing_function(int dim,float *window){
    int I;
    double step,x,y;

    step = 2.0*PI / (double)(dim+1);
    for(i=0,x=step;i<dim;i++,x+=step){
        y = (1.0 - cos(x))/2.0;
        window[i] = (float)sqrt(y);
    }

return(1);
}

int window_1d_vector(
    float *array,
    int data_length,
    int full_length
){
    int i;
    float *parray,*pwindow;

    float *window_function = new float[data_length];
    load_windowing_function(data_length,window_function);
    parray = array;
    pwindow = window_function;
    for(i=0;i<data_length;i++)*(parray++) *= *(pwindow++);
    if(full_length != data_length){
        for(i=0;i<(full_length - data_length);i++)*(parray++) = (float)0.0;
    }
    delete [] window_function;
    return(1);
}


// this module specifically designed for the rough thumbnail registration
/* in an earlier version of this routine, I performed bi-linear interpolatioon
on the pixels, but now think this is overkill becauseof the later refinement
anyway, who knows */
int rotate_scale_translate_image(
    float *out,
    int outdim,
    float *in,
    int inxdim,
    int inydim,
    int orig_xdim,
    int orig_ydim,
    int downsample,
    float rotation,
    float scale
){
    int i,j,xx,yy;
    float a_const,b_const,x,y,dx,dy,*pout;
    float middle_in_x, middle_in_y,middle_out;

    /* make sure to place the center of the original array at the center of
    the output array;  this helps later translation bookkeeping */
    middle_in_x = (float)(orig_xdim - downsample)/(float)downsample/(float)2.0;
    middle_in_y = (float)(orig_ydim - downsample)/(float)downsample/(float)2.0;
    middle_out = (float)(outdim-1)/(float)2.0;
    rotation = -rotation;  // who can keep track of CW and CCW anyway???
    a_const = (float)cos((double)rotation*PI/180.0)*scale;
    b_const = (float)sin((double)rotation*PI/180.0)*scale;
    dx = a_const;
    dy = b_const;
    pout = out;
    for(i=0;i<outdim;i++){
        x = middle_in_x - a_const*middle_out + b_const*(middle_out-(float)i) + (float)0.5;
        y = middle_in_y - b_const*middle_out - a_const*(middle_out-(float)i) + (float)0.5;
```

```
    for(j=0;j<outdim;j++){
        if(x<(float)0.5 || x>(float)(inxdim-1)+(float)0.5||y<(float)0.5 ||
            y>(float)(inydim-1)+(float)0.5)*(pout++)=(float)0.0;
        else {
            xx = (int)x;
            yy = (int)y;
            *(pout++) = in[yy*outdim+xx];
        }
        x+=dx;
        y+=dy;
    }
  }

return(1);
}



int gmf(
  float *real1,
  float *real2,
  int dim,
  int bits,
  int number_candidates,
  float *x_offset,
  float *y_offset,
  float *value,
  int type
){
  int dim2 = dim/2,i,j,k,l,ok,jtemp,ktemp;
  int x_off[MAX_CANDIDATES],y_off[MAX_CANDIDATES];
  float mag1,mag2,dot,cross,highest,ratio,ymedian[3],xmedian[3],*py,*px;
  float *preal1,*preal2,*pimaginary1,*pimaginary2;
  float ftmp,dott;

  /* calculate phase differences and reload them into real1 and imaginary1 */
  /* keep phase differences to PI to -PI */
  preal1=real1;pimaginary1=&real1[dim];
  preal2=real2;pimaginary2=&real2[dim];
  for(i=0;i<(1+dim/2);i++){
      for(j=0;j<dim;j++){
          mag1 = (float)sqrt( (double)(*preal1 * *preal1 + *pimaginary1 * *pimaginary1) );
          mag2 = (float)sqrt( (double)(*preal2 * *preal2 + *pimaginary2 * *pimaginary2) );
          if(mag1 == (float)0.0)mag1=(float)SMALL;
          if(mag2 == (float)0.0)mag2=(float)SMALL;
          dot = (*preal1 * *preal2 + *pimaginary1 * *pimaginary2)/mag1/mag2;
          dott = (float)1.0 - dot*dot;
          if(dott<(float)0.0)dott=(float)0.0;
          dott = (float)sqrt( (double)dott );
          cross = *preal1 * *(pimaginary2++) - *(preal2++) * *pimaginary1;
          if(cross < (float)0.0)cross = -(float)1.0;
          else cross = (float)1.0;
          ftmp = mag2;
          dot*=ftmp;dott*=ftmp;
          *(preal1++) = dot;
          *(pimaginary1++) = cross*dott;
      }
      preal1+=dim;
      pimaginary1+=dim;
      preal2+=dim;
      pimaginary2+=dim;
  }

  /* now back into the original domain, then shift the array for simplicity */
  realfft2d_in_place(real1,bits,1,wr,wi);
  shift_array(real1,dim);

  // temporary display results
  //return(1);

  /* then find the top 'candidate' number of points, loading their parameters along the way */
  for(i=0;i<number_candidates;i++){
      highest = -(float)1e20;
      preal1 = real1;
      for(j=0;j<dim;j++){
          for(k=0;k<dim;k++){
              if( *preal1 > highest ){
                  /* check to see if this is within PICK_RADIUS of a previous choice */
                  ok = 1;
                  l = i;
                  while( l-- > 0 ){
                      if( abs(j-y_off[l]) < PICK_RADIUS ||
                         abs(j-dim-y_off[l]) < PICK_RADIUS ||
                         abs(j+dim-y_off[l]) < PICK_RADIUS){
                          if( abs(k-x_off[l]) < PICK_RADIUS ||
                             abs(k+dim-x_off[l]) < PICK_RADIUS ||
                             abs(k-dim-x_off[l]) < PICK_RADIUS )ok=0;
                      }
                  }
                  if(ok){
                      highest = *preal1;
                      x_off[i] = k;
                      y_off[i] = j;
                  }
              }
              preal1++;
          }
      }
  }

  /* step through the found candidates, finding inter-sample values for the peak location */
  for(i=0;i<number_candidates;i++){
      ymedian[0]=ymedian[1]=ymedian[2]=(float)0.0;
      xmedian[0]=xmedian[1]=xmedian[2]=(float)0.0;
      py = ymedian;
      for(j=-1;j<2;j++){
          jtemp = y_off[i]+j;
          if(jtemp < 0)jtemp=dim-1;
          else if(jtemp==dim)jtemp=0;
          px = xmedian;
          for(k=-1;k<2;k++){
              ktemp = x_off[i]+k;
              if(ktemp < 0)ktemp=dim-1;
              else if(ktemp==dim)ktemp=0;
              *py += real1[jtemp*dim+ktemp];
              *(px++) += real1[jtemp*dim+ktemp];
          }
          py++;
      }
      /* now find median values */
      ratio =  get_median_float(ymedian);
      y_offset[i] = (float)dim2 - (  (float)y_off[i] + ratio  );
      ratio =  get_median_float(xmedian);
      x_offset[i] = (float)dim2 - (  (float)x_off[i] + ratio  );
      value[i] = real1[x_off[i] + dim*y_off[i]];
  }

return(1);
}

    // simple sub-routine for direct_registration
int get_working_dimension(
    int alignment_mode,
    int xdim1,
    int ydim1,
    int xdim2,
    int ydim2,
    int *downsample
){
    int highest=xdim1,go=1,fftdim;

    if(ydim1>highest)highest = ydim1;
    if(xdim2>highest)highest = xdim2;
    if(ydim2>highest)highest = ydim2;

    switch(alignment_mode){
        case 0 :   // no downsampling
            *downsample = 1;
            fftdim = 1;
            while( go ){
                if( highest > fftdim ){
                    fftdim*=2;
                }
                else go = 0;
            }
            break;
        case 1:    // nominal downsampling
            *downsample = ((highest-1)/NOMINAL_DOWNSAMPLE_DIM)+1;
            fftdim = NOMINAL_DOWNSAMPLE_DIM;
            break;
        case 2:    // super downsampling
            *downsample = ((highest-1)/SUPER_DOWNSAMPLE_DIM)+1;
            fftdim = SUPER_DOWNSAMPLE_DIM;
            break;
    }

    return(fftdim);
}

// another sub-routine for direct registration
int copy_downsample_window(
    unsigned char *in,
    int xdim,
    int ydim,
    float *out,
    int outdim,
    int downsample
){
    unsigned char *pin;
    int i,j;
```

```
float *pout,*pwindow,normalize;

pin = in;
memset(out,0,outdim*outdim*sizeof(float));
    for(i=0;i<ydim;i++){
    pout = &out[ (i/downsample) * outdim ];
    for(j=0;j<xdim;j++){
        pout[ j/downsample ] += (float)*(pin++);
    }
}

// normalize it for downsampling
if(downsample > 1 ){
    xdim = 1 + (xdim-1)/downsample;
    ydim = 1 + (ydim-1)/downsample;
    normalize = (float)downsample * (float)downsample;
    for(i=0;i<ydim;i++){
        pout = &out[ i * outdim ];
        for(j=0;j<xdim;j++){
            *(pout++) /= normalize;
        }
    }
}

if(WINDOW_ORIGINALS){
    float *window_function = new float[outdim];
    load_windowing_function(xdim,window_function);
    pout = out;
    for(i=0;i<ydim;i++){
        pwindow = window_function;
        for(j=0;j<xdim;j++){
            *(pout++) *= *(pwindow++);
        }
        pout+=(outdim-xdim);
    }
    load_windowing_function(ydim,window_function);
    pout = out;
    for(i=0;i<ydim;i++){
        pwindow = &window_function[i];
        for(j=0;j<xdim;j++){
            *(pout++) *= *pwindow;
        }
        pout+=(outdim-xdim);
    }
    delete [] window_function;
}

return(1);
}


int fourier_mellin_transform(
  float *in,
  float *ftemp,
  int dim,
  float *out
){
  int i,j;
  float *pout,*pwindow;

  convert_to_magnitude(ftemp,in,dim);
  log_polar_remap(ftemp,out,dim);
  if(WINDOW_LOGPOLAR_LOG){
      float *window_function = new float[lp_sampling];
      load_windowing_function(lp_sampling,window_function);
      pout = out;
      for(i=0;i<lp_sampling;i++){
          pwindow = &window_function[i];
          for(j=0;j<lp_sampling;j++){
              *(pout++) *= *pwindow;
          }
      }
      delete [] window_function;
  }

  return(1);
}


int get_best_candidate(
  int number_candidates,
  float *ftemp,
  int dim,
  int bits,
  float *in,
  int xdim,
  int ydim,
  int xdim_orig,
  int ydim_orig,
  int downsample,
          float *rotation,
          float *scale,
          float *x_trans,
          float *y_trans,
          float *template_real
          ){
          int i,highest_i,j;
          float highest = -(float)1e20,xtrans,ytrans,value;

          for(i=0;i<number_candidates;i++){
              for(j=0;j<2;j++){
                  /* rotate and scale suspect_real image into ftemp */
                  rotate_scale_translate_image(ftemp, dim, in,xdim,ydim,xdim_orig,ydim_orig,
                      downsample,rotation[i]+(float)j*(float)180.0,scale[i]);
                  realfft2d_in_place(ftemp,bits,0,wr,wi);
                  gmf(template_real,ftemp,dim,bits, 1, &xtrans, &ytrans, &value, 1);
                  if(value > highest){
                      highest = value;
                      highest_i = i;
                      if(j==1)rotation[i] += (float)180.0;
                      x_trans[i]=xtrans;
                      y_trans[i]=ytrans;
                  }
              }
          }
          rotation[0]=rotation[highest_i];
          scale[0]=scale[highest_i];
          x_trans[0]=x_trans[highest_i];
          y_trans[0]=y_trans[highest_i];

          return(1);
      }

      double log_1D_remap(
          float *in,
          float *out,
          int dim
      ){
          int i,dim2 = dim/2,xx;
          float *pin,*pout;
          double radius,fracx;
          double scale_increment_1d;

          scale_increment_1d=pow( 1.0/(double)START_RADIUS_1D, 1.0/(double)dim);
          pout = out;
          for(i=0;i<dim;i++){
              radius = (START_RADIUS_1D*(double)dim2) * pow(scale_increment_1d,(double)i);
              xx = (int)radius;
              fracx = radius - (double)xx ;
              pin = &in[xx];
              *pout = (float) ( (1.0-fracx) * (double)*(pin++) );
              *(pout++) += (float) ( fracx* (double)*pin );
          }

      return(scale_increment_1d);
      }


      int gmf_1d(
          float *real1,
          float *imaginary1,
          float *real2,
          float *imaginary2,
          int dim,
          int bits,
          float *offset
      ){
          int i,highest_i;
          float *preal1,*preal2,*pimaginary1,*pimaginary2;
          float mag1,mag2,dot,dott,cross,median[3],highest,ratio,ftmp;

          /* calculate phase differences and reload them into real1 and imaginary1 */
          /* keep phase differences to PI to -PI */
          preal1=real1;pimaginary1=imaginary1;
          preal2=real2;pimaginary2=imaginary2;
          for(i=0;i<dim;i++){
              mag1 = (float)sqrt( (double)(*preal1 * *preal1 + *pimaginary1 * *pimaginary1) );
              mag2 = (float)sqrt( (double)(*preal2 * *preal2 + *pimaginary2 * *pimaginary2) );
              if(mag1 == (float)0.0)mag1=(float)SMALL;
              if(mag2 == (float)0.0)mag2=(float)SMALL;
              dot = (*preal1 * *preal2 + *pimaginary1 * *pimaginary2)/mag1/mag2;
              dott = (float)1.0 - dot*dot;
              if(dott<(float)0.0)dott=(float)0.0;
              dott = (float)sqrt( (double)dott );
              cross = *preal1 * *(pimaginary2++) - *(preal2++) * *pimaginary1;
              if(cross < (float)0.0)cross = -(float)1.0;
              else cross = (float)1.0;
              ftmp = mag2;
              dot*=ftmp;dott*=ftmp;
              *(preal1++) = dot;
```

```
        *(pimaginary1++) = cross*dott;
    }

    fft(real1,imaginary1,bits,1,wr,wi,1);

    /* search for highest value, then median find the center */
    highest = -(float)1e20;
    preal1 = real1;
    for(i=0;i<dim;i++){
        if( *preal1 > highest){
            highest = *preal1;
            highest_i = i;
        }
        preal1++;
    }

    if(highest_i == 0){
        median[0]=real1[dim-1];
        median[1]=real1[0];
        median[2]=real1[1];
    }
    else if(highest_i == (dim-1)){
        median[0]=real1[dim-2];
        median[1]=real1[dim-1];
        median[2]=real1[0];
    }
    else {
        median[0]=real1[highest_i-1];
        median[1]=real1[highest_i];
        median[2]=real1[highest_i+1];
    }
    ratio = get_median_float(median);
    *offset = (float)highest_i + ratio ;
    if( *offset > (float)dim/2.0 ) *offset -= (float)dim;

return(1);
}


int refine_axis(
  unsigned char *ttemplate,
  int template_xdim,
  int template_ydim,
  unsigned char *suspect,
  int suspect_xdim,
  int suspect_ydim,
  float *x,
  float *y,
  int which
){
  unsigned char *psuspect;
  int i,j,highest,fftdim,bits,xx,yy,xdim,ydim;
  float x0,x1,x2,y0,y1,y2,*psuspect_integral,*ptemplate_integral;
  float scan_x,scan_y,jump_x,jump_y,current_x,current_y;
  float scale,translation,xdistance,ydistance,suspect_dc,template_dc,frac;
  double scale_increment_1d;

  /* first convert the y axis version to the x axis version */
  x0 = x[0]; y0 = y[0];
  if(which){
      x1 = x[2]; y1 = y[2];
      x2 = x[1]; y2 = y[1];
      xdim = suspect_ydim;
      ydim = suspect_xdim;
  }
  else {
      x1 = x[1]; y1 = y[1];
      x2 = x[2]; y2 = y[2];
      xdim = suspect_xdim;
      ydim = suspect_ydim;
  }

  /* determine the next highest power of two above higher of the two suspect axes */
  if(suspect_xdim > suspect_ydim)highest = suspect_xdim;
  else highest = suspect_ydim;
  bits = 1 + (int)( log( (double)highest - 0.5 ) / log(2.0) );
  fftdim = (int)pow(2.0,(double)bits + 0.00000001);

  float *template_integral = new float[fftdim];
  float *suspect_Integral = new float[fftdim];
  float *template_integral_imaginary = new float[fftdim];
  float *suspect_Integral_Imaginary = new float[fftdim];
  float *template_integral_copy = new float[fftdim];
  float *suspect_Integral_copy = new float[fftdim];

  /* load suspect integral waveform */
  psuspect_integral = suspect_integral;
  for(j=0;j<fftdim;j++)*(psuspect_integral++) = (float)0.0;
  if(!which){
      psuspect = suspect;
          for(i=0;i<suspect_ydim;i++){
              psuspect_integral = suspect_integral;
              for(j=0;j<suspect_xdim;j++)*(psuspect_integral++) += (float)*(psuspect++);
          }
      }
      else {
          psuspect = suspect;
          psuspect_integral = suspect_integral;
          for(i=0;i<suspect_ydim;i++){
              for(j=0;j<suspect_xdim;j++)*psuspect_integral += (float)*(psuspect++);
              psuspect_integral++;
          }
      }
      suspect_dc = (float)0.0;
      psuspect_integral = suspect_integral;
      for(i=0;i<xdim;i++)suspect_dc += *(psuspect_integral++);
      suspect_dc /= (float)xdim;
      psuspect_integral = suspect_integral;
      for(i=0;i<xdim;i++)*(psuspect_integral++)/=suspect_dc;
      memcpy(suspect_integral_copy,suspect_integral,sizeof(float)*fftdim);

      /* calculate scan elements that will be used in following stuff */
      scan_x = (x1-x0)/(float)(xdim-1);
      scan_y = (y1-y0)/(float)(xdim-1);
      jump_x = (x2-x0)/(float)(ydim-1);
      jump_y = (y2-y0)/(float)(ydim-1);
      /* the next routines are split up since the one where the patch (suspect) is
      outside the boundaries of the template forces boundary checking */
      if(x[0]>=0.0 && x[0]<=(float)(template_xdim-1) &&
          x[1]>=0.0 && x[1]<=(float)(template_xdim-1) &&
          x[2]>=0.0 && x[2]<=(float)(template_xdim-1) &&
          x[3]>=0.0 && x[3]<=(float)(template_xdim-1) &&
          y[0]>=0.0 && y[0]<=(float)(template_ydim-1) &&
          y[1]>=0.0 && y[1]<=(float)(template_ydim-1) &&
          y[2]>=0.0 && y[2]<=(float)(template_ydim-1) &&
          y[3]>=0.0 && y[3]<=(float)(template_ydim-1) ){

          ptemplate_integral = template_integral;
          for(j=0;j<fftdim;j++)*(ptemplate_integral++) = (float)0.0;
          for(i=0;i<ydim;i++){
              current_x = x0 + (float)i * jump_x + (float)0.5;  // the addition of 0.5 is simply
rounding
              current_y = y0 + (float)i * jump_y + (float)0.5;
              ptemplate_integral = template_integral;
              for(j=0;j<xdim;j++){
                  xx = (int)current_x;
                  yy = (int)current_y;
                  *(ptemplate_integral++) += (float)ttemplate[yy*template_xdim+xx];
                  current_x += scan_x;
                  current_y += scan_y;
              }
          }
      }
      else {

          ptemplate_integral = template_integral;
          for(j=0;j<fftdim;j++)*(ptemplate_integral++) = (float)0.0;
          for(i=0;i<ydim;i++){
              current_x = x0 + (float)i * jump_x + (float)0.5;  // the addition of 0.5 is simply
rounding
              current_y = y0 + (float)i * jump_y + (float)0.5;
              ptemplate_integral = template_integral;
              for(j=0;j<xdim;j++){
                  xx = (int)current_x;
                  yy = (int)current_y;
                  if(xx<0||xx>=template_xdim||yy<0||yy>=template_ydim)ptemplate_integral++;
                  else *(ptemplate_integral++) += (float)ttemplate[yy*template_xdim+xx];
                  current_x += scan_x;
                  current_y += scan_y;
              }
          }


      }
      template_dc = (float)0.0;
      ptemplate_integral = template_integral;
      for(i=0;i<xdim;i++)template_dc += *(ptemplate_integral++);
      template_dc /= (float)xdim;
      ptemplate_integral = template_integral;
      for(i=0;i<xdim;i++)*(ptemplate_integral++)/=template_dc;
      memcpy(template_integral_copy,template_integral,sizeof(float)*fftdim);

      /* now perform a scale and translation matching of the two integrals */
      window_1d_vector(template_integral,xdim,fftdim);
      window_1d_vector(suspect_Integral,xdim,fftdim);
      memset(suspect_integral_Imaginary,0,sizeof(float)*fftdim);
      memset(template_integral_imaginary,0,sizeof(float)*fftdim);
      fft(suspect_integral,suspect_integral_imaginary,bits,0,wr,wi,1);
      fft(template_integral,template_integral_imaginary,bits,0,wr,wi,1);
      // next routine places output Into _integral array
```

```
    convert_to_magnitude_1d_inplace(suspect_integral,suspect_integral_imaginary,fftdim);
    convert_to_magnitude_1d_inplace(template_integral,template_integral_imaginary,fftdim);
    // next routine places output inot _integral_imaginary array
    scale_increment_1d = log_1D_remap(suspect_integral,suspect_integral_imaginary,fftdim);
    scale_increment_1d = log_1D_remap(template_integral,template_integral_imaginary,fftdim);
    // copy output back into fundamental array and zero out _imaginary
    memcpy(suspect_integral,suspect_integral_imaginary,sizeof(float)*fftdim);
    memcpy(template_integral,template_integral_imaginary,sizeof(float)*fftdim);
    memset(suspect_integral_imaginary,0,sizeof(float)*fftdim);
    memset(template_integral_imaginary,0,sizeof(float)*fftdim);
    // now do the 1d fourier mellin trot
    window_1d_vector(template_integral,fftdim,fftdim);
    window_1d_vector(suspect_integral,fftdim,fftdim);
    fft(suspect_integral,suspect_integral_imaginary,bits,0,wr,wi,1);
    fft(template_integral,template_integral_imaginary,bits,0,wr,wi,1);

    /* gmf_1d to find any small scaling difference between the two */
    gmf_1d(suspect_integral,suspect_integral_imaginary,template_integral,
        template_integral_imaginary,fftdim,bits,&scale);

    //scale *= (float)0.6;  // slight damping factor
    scale = (float)pow(scale_increment_1d,(double)scale);

    // update the x's and y's
    xdistance = (x1-x0);
    xdistance *= ((float)1.0 - scale);
    ydistance = (y1-y0);
    ydistance *= ((float)1.0 - scale);
    x[3] += xdistance; y[3] += ydistance;
    x[4] += xdistance/(float)2.0; y[4] += ydistance/(float)2.0;
    if(which){
        x[2] += xdistance; y[2] += ydistance;
        x1 = x[2]; y1 = y[2];
    }
    else {
        x[1] += xdistance; y[1] += ydistance;
        x1 = x[1]; y1 = y[1];
    }

    /* now with the new scale information, perform a gmf on the original and its rescaled
    counterpart */
    ptemplate_integral = template_integral;
    scale = (float)1.0 / scale;
    float lllast;
    for(i=0,current_x=(float)0.0;i<xdim;i++,current_x+=scale){
        xx = (int)current_x;
        if(xx >= xdim-1)*(ptemplate_integral++) = lllast;
        else {
            frac = current_x - (float)xx;
            *ptemplate_integral = ((float)1.0-frac) * template_integral_copy[xx];
            *(ptemplate_integral++) += frac * template_integral_copy[xx+1];
        }
        lllast = *(ptemplate_integral-1);
    }

    // window the new scaled array; other one should be copy of windowed original
    memcpy(suspect_integral,suspect_integral_copy,sizeof(float)*fftdim);
    window_1d_vector(template_integral,xdim,fftdim);
    window_1d_vector(suspect_integral,xdim,fftdim);
    memset(suspect_integral_imaginary,0,sizeof(float)*fftdim);
    memset(template_integral_imaginary,0,sizeof(float)*fftdim);
    fft(suspect_integral,suspect_integral_imaginary,bits,0,wr,wi,1);
    fft(template_integral,template_integral_imaginary,bits,0,wr,wi,1);

    // now find the translation
    gmf_1d(suspect_integral,suspect_integral_imaginary,template_integral,
        template_integral_imaginary,fftdim,bits,&translation);

    // adjust x and y accordingly
    translation *= (float)0.5;  // I think this accounts for the fact that scaling has changed
origins????? very kludge
    scan_x *= translation;
    scan_y *= translation;
    x[0] += scan_x; y[0] += scan_y;
    x[1] += scan_x; y[1] += scan_y;
    x[2] += scan_x; y[2] += scan_y;
    x[3] += scan_x; y[3] += scan_y;
    x[4] += scan_x; y[4] += scan_y;

    delete [] template_integral;
    delete [] suspect_integral;
    delete [] template_integral_imaginary;
    delete [] suspect_integral_imaginary;
    delete [] template_integral_copy;
    delete [] suspect_integral_copy;

    return(0);
}

float refined_rotation(
        float *x,
        float *y,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        unsigned char *ttemplate,
        int template_xdim,
        int template_ydim
    ){
        int i,xx,yy,count_template,count_suspect;
        float line_integral[REFINED_ROTATION_DIMENSION],*pli,*pli_template;
        float line_integral_template[REFINED_ROTATION_DIMENSION];
        float line_integral_imaginary[REFINED_ROTATION_DIMENSION];
        float line_integral_template_imaginary[REFINED_ROTATION_DIMENSION];
        float angle,x_suspect,y_suspect,x1_suspect,y1_suspect,dx_suspect,dy_suspect;
        float x_template,y_template,x1_template,y1_template,dx_template,dy_template;
        float top_x_suspect=(float)(suspect_xdim-1),top_y_suspect=(float)(suspect_ydim-1);
        float top_x_template=(float)(template_xdim-1),top_y_template=(float)(template_ydim-1);
        float a_const,b_const,tweak,dc_suspect,dc_template;
        float new_x,new_y,yaxis_y,yaxis_x,xaxis_x,xaxis_y;

        yaxis_x = (x[2]-x[0])/(float)(suspect_ydim-1);  /* this gives the unit vector in terms of
    the suspect array */
        yaxis_y = (y[2]-y[0])/(float)(suspect_ydim-1);
        xaxis_x = (x[1]-x[0])/(float)(suspect_xdim-1);
        xaxis_y = (y[1]-y[0])/(float)(suspect_xdim-1);

        /* create line integral sweep around suspect's and template's center point */
        pli = line_integral;
        pli_template = line_integral_template;
        dc_suspect = dc_template=(float)0.0;
        for(i=0;i<REFINED_ROTATION_DIMENSION;i++){
            angle = (float)i * (float)PI / (float)REFINED_ROTATION_DIMENSION;

            x_suspect = x1_suspect = (float)0.5 + top_x_suspect/(float)2.0;
            y_suspect = y1_suspect = (float)0.5 + top_y_suspect/(float)2.0;
            dx_suspect = (float)sin((double)angle);
            dy_suspect = (float)cos((double)angle);
            x_suspect+=dx_suspect;x1_suspect-=dx_suspect;
            y_suspect+=dy_suspect;y1_suspect-=dy_suspect;

            x_template = x1_template = (float)0.5+x[4];
            y_template = y1_template = (float)0.5+y[4];
            dx_template = (xaxis_x*dx_suspect+yaxis_x*dy_suspect);
            dy_template = (xaxis_y*dx_suspect+yaxis_y*dy_suspect);
            x_template+=dx_template;x1_template-=dx_template;
            y_template+=dy_template;y1_template-=dy_template;

            *pli = (float)0.0;
            *pli_template = (float)0.0;
            count_template=0;count_suspect=0;
            while(x_suspect>0.0 && x_suspect<top_x_suspect && y_suspect>0.0 &&
    y_suspect<top_y_suspect){
                xx = (int)x_suspect;
                yy = (int)y_suspect;
                *pli += suspect[yy*suspect_xdim+xx];

                xx = (int)x1_suspect;
                yy = (int)y1_suspect;
                *pli += suspect[yy*suspect_xdim+xx];

                x_suspect+=dx_suspect;x1_suspect-=dx_suspect;
                y_suspect+=dy_suspect;y1_suspect-=dy_suspect;
                count_suspect++;

    if(y_template>0.0&&y_template<top_y_template&&x_template>0.0&&x_template<top_x_template

    &&y1_template>0.0&&y1_template<top_y_template&&x1_template>0.0&&x1_template<top_x_template){
                    xx = (int)x_template;
                    yy = (int)y_template;
                    *pli_template += ttemplate[yy*template_xdim+xx];

                    xx = (int)x1_template;
                    yy = (int)y1_template;
                    *pli_template += ttemplate[yy*template_xdim+xx];

                    x_template+=dx_template;x1_template-=dx_template;
                    y_template+=dy_template;y1_template-=dy_template;
                    count_template++;
                }
            }
            *pli /= (float)count_suspect;
            *pli_template /= (float)count_template;
            dc_suspect += *(pli++);
            dc_template += *(pli_template++);
        }

        /* now one-d fft them and one d gmf */
        memset(line_integral_imaginary,0,sizeof(float)*REFINED_ROTATION_DIMENSION);
```

```
    memset(line_integral_template_imaginary,0,sizeof(float)*REFINED_ROTATION_DIMENSION);
    pli = line_integral;
    pli_template = line_integral_template;
    dc_suspect /= (float)REFINED_ROTATION_DIMENSION;
    dc_template /= (float)REFINED_ROTATION_DIMENSION;
    for(i=0;i<REFINED_ROTATION_DIMENSION;i++){
        *(pli++) -= dc_suspect;
        *(pli_template++) -= dc_template;
    }
    fft(line_integral,line_integral_imaginary,REFINED_ROTATION_BITS,0,wr,wi,1);
    fft(line_integral_template,line_integral_template_imaginary,REFINED_ROTATION_BITS,0,wr,wi,1);


gmf_1d(line_integral,line_integral_imaginary,line_integral_template,line_integral_template_imaginary,
      REFINED_ROTATION_DIMENSION,REFINED_ROTATION_BITS,&tweak);

    tweak *= (float)0.5; // slight damping factor

    tweak *= -((float)180.0/(float)REFINED_ROTATION_DIMENSION);

    /* update xy0 thru xy3 */
    a_const = (float)cos( (double)tweak * PI /180.0 );
    b_const = (float)sin( (double)tweak * PI /180.0 );

    new_x = a_const*(x[4]-x[0]) - b_const*(y[4]-y[0]);
    new_y = b_const*(x[4]-x[0]) + a_const*(y[4]-y[0]);
    x[0] = x[4] - new_x;
    y[0] = y[4] - new_y;
    new_x = a_const*(x[4]-x[1]) - b_const*(y[4]-y[1]);
    new_y = b_const*(x[4]-x[1]) + a_const*(y[4]-y[1]);
    x[1] = x[4] - new_x;
    y[1] = y[4] - new_y;
    new_x = a_const*(x[4]-x[2]) - b_const*(y[4]-y[2]);
    new_y = b_const*(x[4]-x[2]) + a_const*(y[4]-y[2]);
    x[2] = x[4] - new_x;
    y[2] = y[4] - new_y;
    new_x = a_const*(x[4]-x[3]) - b_const*(y[4]-y[3]);
    new_y = b_const*(x[4]-x[3]) + a_const*(y[4]-y[3]);
    x[3] = x[4] - new_x;
    y[3] = y[4] - new_y;

return(tweak);
}

int Align::fine_tune_x_y(unsigned char *ttemplate,
                         int template_xdim,
                         int template_ydim,
                         unsigned char *suspect,
                         int suspect_xdim,
                         int suspect_ydim,
                         float *x,
                         float *y,
                         float *rotation)
{
    //int foo=1;
    float refinement;

    //while(foo){
    /* find xscale, xtrans optimal pair */
    refine_axis(ttemplate,template_xdim,template_ydim,suspect,suspect_xdim,
        suspect_ydim,x,y,0);
    /* find yscale, ytrans optimal pair */
    refine_axis(ttemplate,template_xdim,template_ydim,suspect,suspect_xdim,
        suspect_ydim,x,y,1);
    /* fine tune rotation */
    refinement = refined_rotation(x,y,suspect,suspect_xdim,suspect_ydim,ttemplate,
        template_xdim,template_ydim);
    // NOTE: SOME CONFUSION ABOUT WHETHER NEXT LINE SHOULD BE -= OR +=
    *rotation += refinement;
    //}

    m_alignStatus.refinement = refinement;

    return(1);
}


/* subroutine for direct registration */
int get_corners_and_center(
    float *x,
    float *y,
    float rotation,
    float scale,
    float x_trans,
    float y_trans,
    int xdim,
    int ydim,
    int fftdim,
    int downsample
){
    float a_const,b_const;

    /* the center of the suspect array should translate to...
    (fftdim*downsample - 1)/2.0 - x_trans*downsample, same on y??? */

    /* note that the origin of the downsampled arrays actually is
    positioned at (downsample-1)/2,(downsample-1)/2 in the coordinates of the
    original arrays */
    x_trans *= (float)downsample;
    y_trans *= (float)downsample;

    x[4] = (float)(fftdim*downsample - 1)/(float)2.0 + x_trans;
    y[4] = (float)(fftdim*downsample - 1)/(float)2.0 + y_trans;

    a_const = (float)cos((double)rotation*PI/180.0)/scale;
    b_const = (float)sin((double)rotation*PI/180.0)/scale;

    x[0] = x[4] - (a_const*(float)(xdim-1) - b_const*(float)(ydim-1))/(float)2.0;
    y[0] = y[4] - (b_const*(float)(xdim-1) + a_const*(float)(ydim-1))/(float)2.0;

    x[1] = x[4] + (a_const*(float)(xdim-1) + b_const*(float)(ydim-1))/(float)2.0;
    y[1] = y[4] + (b_const*(float)(xdim-1) - a_const*(float)(ydim-1))/(float)2.0;

    x[2] = x[4] - (a_const*(float)(xdim-1) + b_const*(float)(ydim-1))/(float)2.0;
    y[2] = y[4] - (b_const*(float)(xdim-1) - a_const*(float)(ydim-1))/(float)2.0;

    x[3] = x[4] + (a_const*(float)(xdim-1) - b_const*(float)(ydim-1))/(float)2.0;
    y[3] = y[4] + (b_const*(float)(xdim-1) + a_const*(float)(ydim-1))/(float)2.0;


return(1);
}

int final_image(
    unsigned char *out,
    int outxdim,
    int outydim,
    unsigned char *in,
    int inxdim,
    int inydim,
    float *x,
    float *y,
    int num_channels,
    int option
){
    unsigned char *pout;
    int i,j,xx,yy;
    float ii,current_x,current_y,fracx,fracy,ftmp,ftmp1,ftmp2,ftmp3,ftmp4;
    float yaxis_x,yaxis_y,xaxis_x,xaxis_y,yaxis_dist,xaxis_dist;
    float x_start,y_start,scan_x,scan_y,jump_x,jump_y;
    unsigned char *pin;

    if(option == 1){  // clear ttemplate array
        pout=out;
        for(i=0;i<(num_channels*outxdim*outydim);i++)*(pout++)=(unsigned char)0;
    }

    yaxis_x = (x[2]-x[0])/(float)(inydim-1);  /* this gives the unit vector in terms of the
suspect array */
    yaxis_y = (y[2]-y[0])/(float)(inydim-1);
    yaxis_dist = (float)sqrt((double)(yaxis_x*yaxis_x+yaxis_y*yaxis_y));
    xaxis_x = (x[1]-x[0])/(float)(inxdim-1);
    xaxis_y = (y[1]-y[0])/(float)(inxdim-1);
    xaxis_dist = (float)sqrt((double)(xaxis_x*xaxis_x+xaxis_y*xaxis_y));

    /* starts is origin dotted with axes */
    x_start = (-x[0] * xaxis_x - y[0] * xaxis_y)/xaxis_dist/xaxis_dist;
    y_start = (-x[0] * yaxis_x - y[0] * yaxis_y)/yaxis_dist/yaxis_dist;
    scan_x = xaxis_x/xaxis_dist/xaxis_dist;
    scan_y = yaxis_x/yaxis_dist/yaxis_dist;
    jump_x = xaxis_y/xaxis_dist/xaxis_dist;
    jump_y = yaxis_y/yaxis_dist/yaxis_dist;

    pout = out;
    for(i=0;i<outydim;i++){
        ii = (float)i;
        current_x = x_start + ii * jump_x;
        current_y = y_start + ii * jump_y;
        if(num_channels==1){
            for(j=0;j<outxdim;j++){
                if(current_x<(float)0.0 || current_x>(float)(inxdim-1)||current_y<(float)0.0
                    || current_y>(float)(inydim-1)){
                    if(option == 0)pout++;  // this option preserves the rest of template
                    else *(pout++) = (unsigned char)0;
                }
                else {
                    xx = (int)current_x;
                    yy = (int)current_y;
                    fracx = current_x - (float)xx ;
                    fracy = current_y - (float)yy ;
```

```
            pin = &in[yy*inxdim + xx];
            ftmp = ((float)1.0-fracx)*((float)1.0-fracy)* (float)*(pin++);
            ftmp += (fracx*((float)1.0-fracy)* (float)*pin);
            pin += (inxdim-1);
            ftmp += ( ((float)1.0-fracx)*fracy* (float)*(pin++) );
            ftmp += ( fracx*fracy * (float)*pin );
            /* debug lines, use with option =0, then it draws a dashed line around suspect
            if(xx == 0 || xx == (inxdim-2) || yy == 0 || yy == (inydim-2))*(pout++)=(unsigned
char)0;
            else *(pout++) = (unsigned char)ftmp;
            */
            *(pout++) = (unsigned char)ftmp;

        }
        current_x += scan_x;
        current_y += scan_y;
      }
    }
    else if(num_channels==3){
      for(j=0;j<outxdim;j++){
        if(current_x<(float)0.0 || current_x>(float)(inxdim-1)||current_y<(float)0.0
          || current_y>(float)(inydim-1)){
            if(option == 0)pout+=3;  // this option preserves the rest of template
            else *(pout++) = *(pout++) = *(pout++) = (unsigned char)0;
        }
        else {
            xx = (int)current_x;
            yy = (int)current_y;
            fracx = current_x - (float)xx ;
            fracy = current_y - (float)yy ;
            ftmp1 = ( (float)1.0 - fracx) * ((float)1.0-fracy );
            ftmp2 = fracx * ((float)1.0-fracy );
            ftmp3 = ( (float)1.0 - fracx) * fracy ;
            ftmp4 = fracx * fracy ;
            pin = &in[3*(yy*inxdim + xx)];
            ftmp = ftmp1 * (float)*pin;
            pin+=3;
            ftmp += (ftmp2 * (float)*pin);
            pin += 3*(inxdim-1);
            ftmp += (ftmp3 * (float)*pin );
            pin+=3;
            ftmp += ( ftmp4 * (float)*pin );
            *(pout++) = (unsigned char)ftmp;
            pin = &in[3*(yy*inxdim + xx)+1];
            ftmp = ftmp1 * (float)*pin;
            pin+=3;
            ftmp += (ftmp2 * (float)*pin);
            pin += 3*(inxdim-1);
            ftmp += (ftmp3 * (float)*pin );
            pin+=3;
            ftmp += ( ftmp4 * (float)*pin );
            *(pout++) = (unsigned char)ftmp;
            pin = &in[3*(yy*inxdim + xx)+2];
            ftmp = ftmp1 * (float)*pin;
            pin+=3;
            ftmp += (ftmp2 * (float)*pin);
            pin += 3*(inxdim-1);
            ftmp += (ftmp3 * (float)*pin );
            pin+=3;
            ftmp += ( ftmp4 * (float)*pin );
            *(pout++) = (unsigned char)ftmp;
        }
        current_x += scan_x;
        current_y += scan_y;
      }
    }
  }
  return(1);
}

// new public subliminal grid stuff, Late April 1996
float *subliminal_grid = new float[130*128];
float *mellin_mag_transform = new float[130*128];
int grid_freq_total = 16;
float *grid_phase = new float[grid_freq_total];
int *grid_x = new int[grid_freq_total];
int *grid_y = new int[grid_freq_total];

int log_polar_remap_public(
  float *in,
  float *out,
  int dim
){
  int i,dim2 = dim/2,xx,yy,j;
  float *pin,*pout;
  double theta,dx,dy,radius[MELLIN_DIMENSION],x,y,fracx,fracy,*pradius;
  int n = MELLIN_DIMENSION;

  double increment= pow( 2.0 , 0.025);
```

```
  double start = sqrt(32.5);
  for(i=0;i<n;i++){
      radius[i] = start * pow(increment,(double)i);
  }
  // pre-filter fourier mag data;
  // first add 90 degree separated  points for 2root2 improvement
  for(i=0;i<64;i++){
      for(j=0;j<64;j++){  // output into left half of original array
          in[63-i+128*j] += in[(1+i)*128+64+j];
      }
  }
  float local_average,*p1,*p2,*p3;
  for(i=0;i<64;i++){
      pout = &in[64*129+i]; // output into right half of original array
      if(i==0)p1 = in;
      else p1 = &in[(i-1)*128];
      p2 = &in[i*128];
      if(i==63)p3 = &in[63*128];
      else p3 = &in[(i+1)*128];
      // first element
      local_average = (*p1 + *(p1+1) + *(p2+1) + *(p3+1) + *p3)/(float)5.0;
      if( *p2 > (float)100.0 * local_average ){
          *pout = (float)100.0;
      }
      else if(local_average < SMALL){
          *pout = SMALL;
      }
      else {
          *pout = *p2 / local_average;
      }
      p1++;p2++;p3++;
      pout -= 128;
      for(j=1;j<63;j++){
          local_average = *(p1-1) + *p1 + *(p1+1) + *(p2+1);
          local_average += *(p3-1) + *p3 + *(p3+1) + *(p2-1);
          local_average /= (float)8.0;
          if( *p2 > (float)100.0 * local_average )*pout = (float)100.0;
          else if(*p2 < SMALL)*pout = SMALL;
          else *pout = *p2 / local_average;
          p1++;p2++;p3++;
          pout -= 128;
      }
      // last element
      local_average = (*p1 + *(p1-1) + *(p2-1) + *(p3-1) + *p3)/(float)5.0;
      if( *p2 > (float)100.0 * local_average )*pout = (float)100.0;
      else if(*p2 < SMALL)*pout = SMALL;
      else *pout = *p2 / local_average;
  }
  // copy horizontal row into vertical column for interp porpoises
  for(i=1;i<64;i++)in[64+i] = in[64+i*128];

  pout = out;
  for(theta=0.0,j=0;j<n; j++,theta += (PI/((double)n)/2.0) ){
      dx = cos(theta);
      dy = sin(theta);
      pradius = radius;
      pout = &out[j];
      for(i=0;i<n;i++){
          x = (double)dim2 + *pradius * dx;
          y = *(pradius++) * dy;
          xx = (int)x;
          yy = (int)y;
          fracx = x - (double)xx ;
          fracy = y - (double)yy ;
          pin = &in[yy*dim + xx];
          *pout = (float) ( (1.0-fracx)*(1.0-fracy)* (double)*(pin++) );
          *pout += (float) ( fracx*(1.0-fracy)* (double)*pin );
          pin += (dim-1);
          *pout += (float) ( (1.0-fracx)*fracy* (double)*(pin++) );
          *pout += (float) ( fracx*fracy * (double)*pin );
          pout += n;
      }
  }

return(1);
}



load_grid_family(
){
    static int done = 0;

    // don't change this without checking its effects on the later grid finding routines
    // such as resolve_orientation
```

```
done = 0;  // force it for now
if(!done){


int i,j,k;

int n = 128;
int n2 = n/2;

// create the spatial domain subliminal grid first

float f0 = (float)0.0;
float real,imag,*pfreq,*pgrid = subliminal_grid;
float grid_scale = (float)3.0;
float i_const = (float)43.245;
float j_const = (float)7.92012;
float k_const = (float)-3.711;
float ki_const = (float)79.0;

// load the standard grid frequencies
// zero it?
for(i=0;i<(n*(n+2));i++){
    subliminal_grid[i]=f0;
}

float *mag_buffer = new float[n*(1+n/2)];
float *pmag = mag_buffer;
for(i=0;i<(n*(1+n/2));i++)*(pmag++) = f0;

int base_x[4] = {8,11,7,9};
int base_y[4] = {1,3,4,7};
int mult = 4,x,y;
int count = 0;
for(j=0;j<4;j++){  // four octal symetry regions (other four are complex conj.; not needed)
    mult = 4;
    for(i=0;i<1;i++){   // the  rings
        for(k=0;k<4;k++){ // now the four bases
            grid_phase[count] = (float)i * i_const + (float)j * j_const + (float)k * k_const;  //
arbitrary for now, to be 'designed'
            grid_phase[count] += ki_const * (float)i * (float)k;
            grid_phase[count] += ki_const * (float)j * (float)k;
            // grid_phase[count] = f0;  // makes for cool patterns
            real = (float)cos( (double)grid_phase[count] );
            imag = (float)sin( (double)grid_phase[count] );
            if(j==0){
                grid_x[count] = -base_x[k] * mult;
                x = n2-base_x[k]*mult;
                y = grid_y[count] = mult * base_y[k];
            }
            else if(j==1){
                grid_x[count] = -base_y[k] * mult;
                x = n2-base_y[k]*mult;
                y = grid_y[count] = mult * base_x[k];
            }
            else if(j==2){
                grid_x[count] = base_y[k] * mult;
                x = n2+base_y[k]*mult;
                y = grid_y[count] = mult * base_x[k];
            }
            else {
                grid_x[count] = base_x[k] * mult;
                x = n2+base_x[k]*mult;
                y = grid_y[count] = mult * base_y[k];
            }
            pfreq = &pgrid[x + 2 * n * y ];
            pmag = &mag_buffer[x+n*y];
            *pfreq = grid_scale * real;
            *(pfreq + n) = grid_scale * imag;
            *pmag = (float)1.0;

            count++;
        }
    }
}

float *wr = new float[n];
float *wi = new float[n];

// fourier transform the dog back into the spatial domain, giving us our standard subliminal grid
realfft2d_in_place(pgrid,7,1,wr,wi );

// now create the final used array in the grid hunt routine
log_polar_remap_public(mag_buffer,mellin_mag_transform,n);

// temporary display results code:  this one has a corresponding return inside hunt_for_grid
//return(1);

// fourier transform the dog into the form it will be used as in the hunt for grid routine
realfft2d_in_place(mellin_mag_transform,7,0,wr,wi );
```

```
        delete [] wr;
        delete [] wi;
        delete [] mag_buffer;
        done = 1;
    }

    return(1);
}


// specific to hunt_for_grid
int add_block_magnitude(
    unsigned char *data,
    float *fourier_mag,
    int n,   // power of 2 dimension of fourier mag
    float *buffer, // needs to be n*(n+2) in length
    int xbumps,
    int ybumps,
    int bump_size,
    int zdim,
    int original_xdim,  // pixel based jump pointer for moving down rows
    int truncated
){

    // load fourier array with bump data
    unsigned char *pdata = data;
    float *pbuffer;
    int i,j;
    for(i=0;i<ybumps;i++){
        pbuffer = &buffer[i*n];
        load_bump_array(
            pbuffer,  // floating point bump array to be filled (output)
            pdata,  // input pixel data
            xbumps,  // number of bumps in this row (not pixels)
            zdim,  // number of channels
            bump_size,  // pixels per bump
            original_xdim - xbumps*bump_size,  // number of raw pixels between
(xdim*bump_size) and entire image array x dimension
            0   // do not overfill the bump buffer
        );
        pdata+=(zdim*original_xdim*bump_size);
    }

    // window it if you please
    /*
    float *window_function = new float[128];
    load_windowing_function(128,window_function);
    float *pwindow_row = window_function;
    float *pwindow_column = window_function;
    pbuffer = buffer;
    for(i=0;i<128;i++){
        pwindow_column = window_function;
        for(j=0;j<128;j++){
            *(pbuffer++) *= *pwindow_row * *(pwindow_column++);
        }
        pwindow_row++;
    }
    delete [] window_function;
    */ // this doesn't seem to help at all!  results seem to get worse


    // fft the dog
    int bits = (int) (log( (double)(n+1) ) / log( 2.0 ) );  // fftdim should always be power
of 2
    realfft2d_in_place(buffer,bits,0,wr,wi);

    // now add its magnitude into the accumulator array
    float *pfourier = fourier_mag;
    float *preal = buffer;
    float *pimag = &buffer[n];
    for(i=0;i<(n/2+1);i++){
        for(j=0;j<n;j++){
            // consider a "cheaty" version of the following, just add the absolute mags,
forget the sqrt
            *pfourier += (float)sqrt(*preal * *preal + *pimag * *pimag);
            preal++;pimag++;pfourier++;
        }
        preal += n;
        pimag += n;
    }

    return(1);
}


int rotate_scale_image(
    unsigned char *data,
```

```
    int xdim,
    int ydim,
    int zdim,
    int bump_size,
    int n,
    int original_xdim,
    float rotation,
    float scale,
    float *out
){
    int n2 = n/2;
    float outcenter = (float)(n-1) / (float)2.0;
    float inxcenter = (float)(xdim-1) / (float)2.0;
    float inycenter = (float)(ydim-1) / (float)2.0;

    // create buffer for input data
    float *buffer = new float[xdim*ydim];

    // load buffer array with bump data
    unsigned char *pdata = data;
    float *pbuffer;
    int i;
    for(i=0;i<ydim;i++){
        pbuffer = &buffer[i*n];
        load_bump_array(
            pbuffer,  // floating point bump array to be filled (output)
            pdata,  // input pixel data
            xdim,  // number of bumps in this row (not pixels)
            zdim,  // number of channels
            bump_size,  // pixels per bump
            original_xdim - xdim*bump_size,  // number of raw pixels between (xdim*bump_size) and entire
image array x dimension
            0   // do not overfill the bump buffer
        );
        pdata+=(zdim*original_xdim*bump_size);
    }

    // now rotate and scale the input image inside buffer, into the output image
    // use xdim/2 and ydim/2 as the center of rotation for the input image
    // use n/2 and n/2 as the center of the output array
    scale = (float)1.0 / scale;
    rotation = -rotation;
    float costheta = scale * (float)cos( (double) rotation * PI / 180.0 );
    float sintheta = scale * (float)sin( (double) rotation * PI / 180.0 );
    float ii,jj,fracx,fracy,*pout = out,*pin,x,y;
    int xx,yy,j;
    for(i=0;i<n;i++){
        ii = (float)i - outcenter;
        for(j=0;j<n;j++){
            jj = (float)j-outcenter;
            x = jj * costheta + ii * sintheta;
            y = ii * costheta - jj * sintheta;
            x+=inxcenter;
            y+=inycenter;
            xx = (int)x;
            yy = (int)y;
            if(xx < 0){
                xx = 0;
                fracx = (float)0.0;
            }
            else if(xx >= xdim-1){
                xx = xdim-2;
                fracx = (float)1.0;
            }
            else fracx = x - (float)xx;
            if(yy < 0){
                yy = 0;
                fracy = (float)0.0;
            }
            else if(yy >= ydim-1){
                yy = ydim-2;
                fracy = (float)1.0;
            }
            else fracy = y - (float)yy;

            pin = &buffer[yy*n + xx];
            *pout = ( ((float)1.0-fracx)*((float)1.0-fracy)* *(pin++) );
            *pout += ( fracx*((float)1.0-fracy)* *pin );
            pin += (n-1);
            *pout += ( ((float)1.0-fracx)*fracy* *(pin++) );
            *(pout++) += ( fracx*fracy * *pin );
        }
    }

    delete [] buffer;

    return(1);
}
```

```
/* this is a specialized function simply meant to find out which of 4
90 degree orientations is the true orientation of the subliminal grid;
the fourier-mellin transform, combined with our "folding" of frequencies,
gives this ambiguity in the first place
*/
int resolve_orientation(
    unsigned char *data,
    int xdim,
    int ydim,
    int zdim,
    int bump_size,
    int n,  // power of 2 used in inverse fft's
    int original_xdim,
    float *rotation,
    float *scale
){

    int mult = 1;
    if(*scale > (float)1.25){  // up n to the next higher power of two
        n*=2;
        mult = 2;
    }

    float *buffer = new float[n*(n+2)];
    int n2 = n/2,i,j;

    rotate_scale_image(
        data,
        xdim,
        ydim,
        zdim,
        bump_size,
        n,
        original_xdim,
        *rotation,
        *scale,
        buffer
    );

    // fft the thing
    int bits = (int) (log( (double)(n+1) ) / log( 2.0 ) );  // fftdim should always be power
of 2
    realfft2d_in_place(buffer,bits,0,wr,wi);  // ultimately, direct calculation may be faster
assuming frequency points < bits*bits

    // save the original phase values
    float *real = new float[grid_freq_total];
    float *imag = new float[grid_freq_total];
    for(i=0;i<grid_freq_total;i++){
        real[i] = buffer[n2 + mult*grid_x[i] + 2*n*mult*grid_y[i]];
        imag[i] = -buffer[n + n2 + mult*grid_x[i] + 2*n*mult*grid_y[i]];
    }

    // now step through the four possible orientations, finding the best fit
    // the current incarnation of this routine is intimately tied to
    // the function load_grid_family
    float highest,high = (float)-1e20,grid_real,grid_imag;
    int highi,tmp;
    float value[4],x_offset[4],y_offset[4];
    for(i=0;i<4;i++){
        // zero out buffer
        memset(buffer,0,sizeof(float)*n*(n+2));
        // multiply this orientation by saved phases
        for(j=0;j<grid_freq_total;j++){
            if(i==0){
                grid_real = (float)cos((double)grid_phase[j]);
                grid_imag = (float)sin((double)grid_phase[j]);
            }
            else if(i==1){
                tmp = (j+grid_freq_total/2)%grid_freq_total;
                grid_real = (float)cos((double)grid_phase[tmp]);
                if(tmp >= grid_freq_total/2)grid_imag = (float)sin((double)grid_phase[tmp]);
                else grid_imag = -(float)sin((double)grid_phase[tmp]);
            }
            else if(i==2){
                grid_real = (float)cos((double)grid_phase[j]);
                grid_imag = -(float)sin((double)grid_phase[j]);
            }
            else {
                tmp = (j+grid_freq_total/2)%grid_freq_total;
                grid_real = (float)cos((double)grid_phase[tmp]);
                if(tmp >= grid_freq_total/2)grid_imag = -(float)sin((double)grid_phase[tmp]);
                else grid_imag = (float)sin((double)grid_phase[tmp]);
            }
            buffer[n2 + mult*grid_x[j] + 2*n*mult*grid_y[j]] = real[j] * grid_real -
imag[j]*grid_imag;
            buffer[n + n2 + mult*grid_x[j] + 2*n*mult*grid_y[j]] = real[j] * grid_imag +
imag[j]*grid_real;
```

```
        }
        realfft2d_in_place(buffer,bits,1,wr,wi);  // ultimately, direct calculation may be faster
 assuming frequency points < bits*bits

        // find highest point
        highest = (float)-1e20;
        float *pbuffer = buffer;
        int high_x,high_y;
        for(j=0;j<(n*n);j++){
            if( *pbuffer > highest ){
                highest = *pbuffer;
                high_y = j/n;
                high_x = j - high_y;
            }
            pbuffer++;
        }

        // load its median inter-sample value
        value[i] = get_2D_median(buffer,n,n,high_x,high_y,&x_offset[i],&y_offset[i]);

        // then, find the highest of the four
        if(highest > high){
            highi = i;
            high = highest;
        }
    }

    // update rotation
    *rotation += (float)highi * (float)90.0;

    delete [] real;
    delete [] imag;

    return(1);
}


/*
This function performs two basic services;  first, it simply attempts
to deteremine if a public subliminal grid exists or not;
if one does exist, then the second basic service is to determine the
rough scale and rotation state of that grid.

The mode_flag variable provides options for how fast v. thorough the algorithms
are.

*/
int hunt_for_grid(
  unsigned char *data,  // input image, unknown signature status
  int xdim,  // its full pixel dimension in x
  int ydim,  // ditto in y
  int zdim,  // number of channels
  int probable_bump_size,  // this is a tricky one to start;  to best function,
      // we will need to specify or "recommend" some standard bumps-per-inch
      // and first look for the signatures in that region
  int total_blocks,  // how hard do we look
  int *present,
  float *scale,
  float *rotation,
  float *mellin_mag_transform
){
  int xblocks,yblocks,i,j,xlength,ylength;
  unsigned char *pdata;

  // the checking takes the first N 128by128 bump regions, FFT's them,
  // converts them to magnitudes, adds them all, then does
  // the fourier \-mellin  check between the added versions and
  // the master public grid FM profile.
  // A Yes/No is generated based on the S/N found between a peak and the
  // background

  // find and use full integral blocks only, unless the data is shorter
  // than a full integral block
  int xbumpsize = xdim/probable_bump_size;
  int ybumpsize = ydim/probable_bump_size;
  xblocks = xbumpsize / SIGNATURE_BLOCK_DIMENSION;  // if 0, doesn't even cover one block but will
still function
  yblocks = ybumpsize / SIGNATURE_BLOCK_DIMENSION;

  // temporary
  total_blocks = xblocks * yblocks;  // again, 0 will function

  // create the basic fourier magnitude array  (SIGDIM*(SIGDIM/2+1)) or 128 by 65
  int n=SIGNATURE_BLOCK_DIMENSION;
  float *fourier_mag = new float[n*(1+n/2)]; // only stores the magnitude
  float *buffer = new float[n*(n+2)]; // give it a full array for processing inside 'add_block'
  int m = MELLIN_DIMENSION;
  float *mellin_mag = new float[m*(m+2)];
  float f0 = (float)0.0;
  for(i=0;i<(n*(1+n/2));i++)fourier_mag[i]=f0;
  int count = 0;
  int truncated;
  for(i=0;i<yblocks;i++){
      for(j=0;j<xblocks;j++){
          count++;
          pdata = &data[(i*xdim+j)*n*probable_bump_size];  // offset to this block
          if(xblocks == 0 || yblocks == 0){
              truncated = 1;
              if(xblocks==0)xlength = xbumpsize;
              else xlength = n;
              if(yblocks==0)ylength = ybumpsize;
              else ylength = n;
          }
          else {
              truncated = 0;
              xlength = n;
              ylength = n;
          }
          add_block_magnitude(
              pdata,
              fourier_mag,
              n,
              buffer,
              xlength,
              ylength,
              probable_bump_size,
              zdim,
              xdim,  // pixel based jump pointer for moving down rows
              truncated
          );
          if(count >= total_blocks){j=xblocks;i=yblocks;}//this kicks it out
      }
  }

  // temporary: ship this one back for display
  // use atemp.bmp as input alignment template file
  //memcpy(mellin_mag_transform,fourier_mag,sizeof(float)*n*(n/2+1));
  //return(1);

  // now fourier mellinize the magnitude profile
  log_polar_remap_public(fourier_mag,mellin_mag,n);
  // temporary display results code
  // use atemp128.bmp as input alignment template file
  //memcpy(mellin_mag_transform,mellin_mag,sizeof(float)*n*n);
  //return(1);

  // fourier transform the dog
  realfft2d_in_place(mellin_mag,7,0,wr,wi );

  load_grid_family();  // will immediately return if already done
  // temporary display results code:  this one has a corresponding return inside
load_grid_family
  //memcpy(mellin_mag_transform,subliminal_grid,sizeof(float)*128*128);
  //return(1);

  // now compare the patterns
  int bits = (int) (log( (double)(n+1) ) / log( 2.0 ) );  // fftdim should always be power
of 2
  int number_candidates = 20;
  float *rotation_buf = new float[number_candidates];
  float *scale_buf = new float[number_candidates];
  float *value = new float[number_candidates];

gmf(mellin_mag,mellin_mag_transform,n,bits,number_candidates,rotation_buf,scale_buf,value,0);

  // temporary display results; matching return in gmf function
  //return(1);

  // a first crack at deciding whether or not a signature/grid is present is possible
  // at this point:  the ratio between value0 and valueN should be above some
  // threshold.  If this is unreliable, then complete the alingment/read process,
  // read the control bits and their checksums, and see if the checksums are right;
  // this will obviously take a longer time to make a negative decision.

  delete [] fourier_mag;
  delete [] buffer;
  delete [] mellin_mag;

  float detection_value = value[0] / value[19];
  float threshold_detect = (float)2.0;  // where's our empirical data anyway, false-positive
curves, true double entendre negatives, etc.
  if(detection_value > threshold_detect){ // we haave a winna
      // if the suspect image has been rotated clockwise, rotation_buf will be positive
      // if the suspect image has been expanded, scale will come back negative
      rotation_buf[0] *= (float)(90.0 / 128.0);
      double increment= pow( 2.0 , 0.025);
```

```
    scale_buf[0] = (float)pow(increment,(double)scale_buf[0]);

    if(xblocks == 0 || yblocks == 0){
        truncated = 1;
        if(xblocks==0)xlength = xbumpsize;
        else xlength = n;
        if(yblocks==0)ylength = ybumpsize;
        else ylength = n;
    }

    // resolve 90 degree ambiguity in rotation/orientation
    resolve_orientation(data,xlength,ylength,zdim,probable_bump_size,
        n,xdim,&rotation_buf[0],&scale_buf[0]);

    *rotation = rotation_buf[0];
    *scale = scale_buf[0];
    *present = 1;


    //now find precise global alignment parameters


  }
  else {  // send back no go on first detect, then get options for quittting or looking harder
    *present = 0;
  }



  delete [] rotation_buf;
  delete [] scale_buf;
  delete [] value;

  return(1);
}

int expiriment(
  unsigned char *data,
  int n
){

  float *imag = new float[n*n];

  //for(i=0;i<n*n;i++)imag[i]=(float)0.0;

  load_grid_family();  // will immediately return if already done

      realfft2d_in_place(subliminal_grid,7,0,wr,wi );

  fft2d(subliminal_grid,imag,7,0,wr,wi );

  return(1);
}



/* main registration program:  to be used as main module inside other programs */
int Align::direct_registration (
  unsigned char *ttemplate,
  int template_xdim,
  int template_ydim,
  unsigned char *suspect,
  int suspect_xdim,
  int suspect_ydim,
  int num_channels
){
  if(1){


      //expiriment(ttemplate,template_xdim);
      //return(1);

      int present;
      float rotation,scale;
      extern float *mellin_mag_transform;
      hunt_for_grid(
          suspect,
          suspect_xdim,
          suspect_ydim,
          num_channels,
          1,
          10,
          &present,
          &scale,
          &rotation,
          mellin_mag_transform
      );

      // temporary:  place mellin_mag_transform into ttemplate for return

          // use atemp.bmp as input alignment template file
          /*
          float highest=(float)-1e20,lowest=(float)1e20;
          int i,n=128;
          for(i=0;i<(n*(n/2+1));i++){
              if( (i/128 < 6) && (abs((i%128)-64) < 6) );
              else {
                  if(mellin_mag_transform[i]>highest)highest=mellin_mag_transform[i];
                  if(mellin_mag_transform[i]<lowest)lowest=mellin_mag_transform[i];
              }
          }
          highest = (float)255.0/(highest-lowest);
          for(i=0;i<(n*(n/2+1));i++){
              if( (i/128 < 6) && (abs((i%128)-64) < 6) )ttemplate[i] = (unsigned char)100;
              else ttemplate[i] = (unsigned char)( (mellin_mag_transform[i] - lowest )*highest);
          }
          */
          // use atemp128.bmp as input alignment template file
          float highest=(float)-1e20,lowest=(float)1e20;
          int i,n=128;
          for(i=0;i<(n*n);i++){
              if(mellin_mag_transform[i]>highest)highest=mellin_mag_transform[i];
              if(mellin_mag_transform[i]<lowest)lowest=mellin_mag_transform[i];
          }
          highest = (float)255.0/(highest-lowest);
          for(i=0;i<(n*n);i++){
              ttemplate[i] = (unsigned char)( (mellin_mag_transform[i] - lowest )*highest);
          }


      }
  else {


      int i,fftdim,bits,array_size,lp_array_size;
      int alignment_mode=2,downsample;
      int number_candidates = MAX_CANDIDATES;     /* number of peaks looked at */
      float rotation[MAX_CANDIDATES],scale[MAX_CANDIDATES],value[MAX_CANDIDATES];
      float x_trans[MAX_CANDIDATES],y_trans[MAX_CANDIDATES],x[5],y[5];
      unsigned char *suspect_lum = new unsigned char[suspect_xdim*suspect_ydim];
      unsigned char *template_lum = new unsigned char[template_xdim*template_ydim];

      // if color image, then create collapse template into a single image,
      // while the real suspect is used during final resampling
      if(num_channels == 3){
          unsigned char *pin,*ptemplate;
          ptemplate =  template_lum;
          pin =  ttemplate+1;
          for(i=0;i<(template_xdim*template_ydim);i++){
              *(ptemplate++) = *pin;  // no need for extreme accuracy
              pin+=3;
          }
          ptemplate =  suspect_lum;
          pin =  suspect+1;
          for(i=0;i<(suspect_xdim*suspect_ydim);i++){
              *(ptemplate++) = *pin;  // no need for extreme accuracy
              pin+=3;
          }
      }

      // find working array size after downsampling (if downsampling is called at all)
      fftdim = get_working_dimension(alignment_mode,template_xdim,template_ydim,
          suspect_xdim,suspect_ydim,&downsample);
      array_size = fftdim*(fftdim+2);
      lp_array_size = lp_sampling*(lp_sampling+2);  // the extra 2 is due to the fft routine
being used
      bits = (int) (log( (double)(fftdim+1) ) / log( 2.0 ) );  // fftdim should always be power
of 2

      // create the requisite arrays
      float *template_real = new float[array_size];
      float *template_lp_real = new float(lp_array_size);
      float *suspect_real = new float[array_size];
      float *suspect_lp_real = new float[lp_array_size];
      float *ftemp = new float[array_size];
      float *suspect_copy = new float[array_size];

      // copy the two inputs into the arrays, with any downsampling and windowing applied
      if(num_channels == 1){
          copy_downsample_window(suspect,suspect_xdim,suspect_ydim,suspect_real,
              fftdim,downsample);
          copy_downsample_window(ttemplate,template_xdim,template_ydim,template_real,
              fftdim,downsample);
      }
      else if(num_channels == 3){
          copy_downsample_window(suspect_lum,suspect_xdim,suspect_ydim,suspect_real,
              fftdim,downsample);
          copy_downsample_window(template_lum,template_xdim,template_ydim,template_real,
```

```
        fftdim,downsample);
    }
    memcpy(suspect_copy,suspect_real,array_size*sizeof(float) );

    /* real-valued 2D FFT both suspect and template into it's half-plane complex */
    realfft2d_in_place(template_real,bits,0,wr,wi);
    realfft2d_in_place(suspect_real,bits,0,wr,wi);

    // calculate fourier mellin transform
    fourier_mellin_transform(template_real,ftemp,fftdim,template_lp_real);
    fourier_mellin_transform(suspect_real,ftemp,fftdim,suspect_lp_real);

    /* assuming the inputs are both real only, then real 2D FFT each */
    realfft2d_in_place(template_lp_real,lp_bits,0,wr,wi);
    realfft2d_in_place(suspect_lp_real,lp_bits,0,wr,wi);

    /* perform generalized matched filter on the two resulting arrays, outputting some number of
    likely candidates, with their associated parameters */
    gmf(template_lp_real,suspect_lp_real,lp_sampling,lp_bits,number_candidates,
         rotation, scale, value, 0);

    // change units on rotation and scale for later stages
    for(i=0;i<number_candidates;i++){
        rotation[i] *= ((float)180.0 / (float)lp_sampling);  // converts to degrees
        scale[i] = (float)pow((double)scale_increment,(double)scale[i]);  // converts to linear scale
    }

    /* now we have a series of candidates ( or 1, and we just need to get the rotation
    and translation information ) wherein one of them should be
    the correct one;  this next routine sifts through all candidates, including both
    the nominal rotation state and the state 180 degrees rotated from the nominal, and
    finds which rotation, scale, and translation gives the highest matched filter
    output; which then will be passed to the last fine tuning stage*/
    // returns best candidate in first element of rotation, scale, x_trans, y_trans
    get_best_candidate(number_candidates,ftemp,fftdim,bits,suspect_copy,
        1+(suspect_xdim-1)/downsample,1+(suspect_ydim-1)/downsample,suspect_xdim,
        suspect_ydim,downsample,rotation,scale,x_trans,y_trans,template_real);

    /* convert the scale/rotation/translation parameters of the downsampled arrays
    into the x and y positions of the four corners of the suspect array, as projected
    onto the template array.  Precision in keeping track of the various coordinate systems
    translates into final alignments to well better than a single pixel, especially
    in light of the subtleties involved with downsampling.  The four corners
    are labelled 0 through 3 in the arrays x and y, where element 0 is the upper left corner
    of the suspect, element 1 is the upper right, element 2 lower left, element 3 lower right.
    The master 0,0 origin is placed at the upper left of the template array, while
    the centerpoints of the two arrays play a role in rotations.  The fifth
    point in the x and y arrays is the centerpoint, used just so you don't have to
    recalculate it all the time*/
    get_corners_and_center(x,y,rotation[0],scale[0],x_trans[0],y_trans[0],
        suspect_xdim,suspect_ydim,fftdim,downsample);

    /* now fine tune the result using tricky tricks, see notebook of Nov 28, 1995 */
    if(num_channels == 1){
        for(i=0;i<100;i++){
            fine_tune_x_y(ttemplate,template_xdim,template_ydim,suspect,suspect_xdim,
                suspect_ydim,x,y,rotation);
        }
    }
    else if(num_channels == 3){
        fine_tune_x_y(template_lum,template_xdim,template_ydim,suspect_lum,suspect_xdim,
            suspect_ydim,x,y,rotation);
    }

    /* last but not least, create the output image array, with various options */
    final_image(ttemplate,template_xdim,template_ydim,suspect,suspect_xdim,
        suspect_ydim,x,y,num_channels,1); // '1' stands for aligned suspect with black everywhere else

    /* Record some results of the alignment process in our status structure */
    m_alignStatus.rotation = rotation[0];
    m_alignStatus.x_scale = scale[0];
    m_alignStatus.y_scale = scale[0];
    m_alignStatus.x_trans = x_trans[0];
    m_alignStatus.y_trans = y_trans[0];

    /* free em all */
    delete [] template_real;
    delete [] template_lp_real;
    delete [] suspect_real;
    delete [] suspect_lp_real;
    delete [] ftemp;
    delete [] suspect_copy;
    delete [] suspect_lum;
    delete [] template_lum;
}

    return(1);
}

/* shell to at least get the main registration program up and running, tested */
#ifdef NEED_MAIN

////////////////////////////////////////////////////////////////////////
// main()
//
// For Geoff's testing purposes, this main() function was used to
// create a stand-alone program which excercised the alignment
// algorithms.  This is #ifdef'd out for the windows version.
////////////////////////////////////////////////////////////////////////
main( int argc, char *argv[] )
{
    int template_xdim,template_ydim,suspect_xdim,suspect_ydim;
    char template_filename[80],suspect_filename[80];
    FILE *inf;

    printf("\nTemplate file name please:  ");
    scanf("%s",&template_filename);
    printf("\nX dimension and Y dimension of template file: ");
    scanf("%d %d",&template_xdim,&template_ydim);
    printf("\nSuspect file name please:  ");
    scanf("%s",&suspect_filename);
    printf("\nX dimension and Y dimension of suspect file: ");
    scanf("%d %d",&suspect_xdim,&suspect_ydim);

    unsigned char *img = new unsigned char[template_xdim*template_ydim*sizeof(unsigned char)];
    unsigned char *img1 = new unsigned char[suspect_xdim*suspect_ydim*sizeof(unsigned char)];

    /* read in binary data into template */
    inf = fopen(template_filename,"rb");
    if(!inf) {
        fprintf(stderr,"register: can't open %s\n",template_filename);
        exit(1);
    }
    fread(img,sizeof(unsigned char),template_xdim*template_ydim,inf);
    fclose(inf);

    inf = fopen(suspect_filename,"rb");
    if(!inf) {
        fprintf(stderr,"register: can't open %s\n",suspect_filename);
        exit(1);
    }
    fread(img1,sizeof(unsigned char),suspect_xdim*suspect_ydim,inf);
    fclose(inf);

    /* returns registered image inside array 'template' */
    direct_registration(img,template_xdim,template_ydim,img1,suspect_xdim,suspect_ydim);

    /*write out  binary data from template */
    inf = fopen("reg_out","wb");
    if(!inf) {
        fprintf(stderr,"register: can't open %s\n","reg_out");
        exit(1);
    }
    fwrite(img,sizeof(unsigned char),template_xdim*template_ydim,inf);
    fclose(inf);

    /* free and clean up */
    delete [] img;
    delete [] img1;

    return(0);
}

#endif //NEED_MAIN
```

ALIGN.H

```
////////////////////////////////////////////////////////////////////////
// FILE: Align.h
//
// DESCRIPTION:
// Header file for the Alignment core algorithm code and the "Align"
// class used to encapsulate this code.
//
// The Alignment code is equivalent to Geoff Rhoads "Register" core
// algorithms, which were first created and run as a stand-alone C program
// on the SGI, then ported to Win95 and Visual C++ as a "console" program,
// and finally incorporated into the Signer windows application.
//
// Copyright (C) 1996 Digimarc Incorporated, all rights reserved.
////////////////////////////////////////////////////////////////////////
#ifndef ALIGN_H
```

```
#define ALIGN_H

// A structure used to define results of the alignment process.
typedef struct
{
  float   rotation;
  float   x_scale;
  float   y_scale;
  float   x_trans;
  float   y_trans;
  float   refinement;
} AlignStatus;

// Function prototypes: entry functions
class Align
{
public:
  Align();
  int direct_registration(unsigned char *ttemplate,
                          int template_xdim,
                          int template_ydim,
                          unsigned char *suspect,
                          int suspect_xdim,
                          int suspect_ydim,
                          int num_channels);
  // Accessor for status
  const AlignStatus GetAlignStatus(void) const {return m_alignStatus;}

private:
  // Private structure which contains results of alignment
  AlignStatus     m_alignStatus;

  int fine_tune_x_y(unsigned char *ttemplate,
                    int template_xdim,
                    int template_ydim,
                    unsigned char *suspect,
                    int suspect_xdim,
                    int suspect_ydim,
                    float *x,
                    float *y,
                    float *rotation);

};


// Function prototypes: private functions
int gmf_1d(float  *real1,
        float    *imaginary1,
        float    *real2,
        float    *imaginary2,
        int      dim,
        int      bits,
        float    *offset);


#endif    // ALIGN_H
```

ALIGNDLG.CPP

```
// AlignDlg.cpp : implementation file
//

#include "stdafx.h"
#include "signer.h"
#include "AlignDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// AlignDlg

IMPLEMENT_DYNAMIC(AlignDlg, CFileDialog)

AlignDlg::AlignDlg(BOOL bOpenFileDialog, LPCTSTR lpszDefExt, LPCTSTR lpszFileName,
      DWORD dwFlags, LPCTSTR lpszFilter, CWnd* pParentWnd) :
      CFileDialog(bOpenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter, pParentWnd)
{
}


BEGIN_MESSAGE_MAP(AlignDlg, CFileDialog)
  //{{AFX_MSG_MAP(AlignDlg)
      // NOTE - the ClassWizard will add and remove mapping macros here.
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

ALIGNDLG.H

```
// AlignDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// AlignDlg dialog

class AlignDlg : public CFileDialog
{
    DECLARE_DYNAMIC(AlignDlg)

public:
    AlignDlg(BOOL bOpenFileDialog, // TRUE for FileOpen, FALSE for FileSaveAs
        LPCTSTR lpszDefExt = NULL,
        LPCTSTR lpszFileName = NULL,
        DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
        LPCTSTR lpszFilter = NULL,
        CWnd* pParentWnd = NULL);

protected:
    //{{AFX_MSG(AlignDlg)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

COXKEY.CPP

```
/////////////////////////////////////////////////////////////////////////////
// FILE: CoXKey.cpp
//
// DESCRIPTION:
// Contains the implementation of the CoExtensive Key class (CoXKey).
// A Coextensive key is also known as a "snowy image" or "code pattern".
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
/////////////////////////////////////////////////////////////////////////////
#include "CoxKey.h"
#include "dibapi.h"


/////////////////////////////////////////////////////////////////////////////
// CoXKey()
//
// The constructor for the class takes a user_key as the seed to the
// random number generator, a pointer to the Bitmapinfo structure
// which defines the dimensions, etc, of the DIB, and a pointer to
// the DIB image space where we will put the snow.  We basically
// seed the random number generator and fill the image data space
// with random values.
// Note that we must be careful to adhere to the core algorithms
// standard that the origin of an image is at the top left.  Since
// Windows Bitmap images (DIBS) usually use the lower left as the
// origin, we need to be careful of the ordering and in the typical
// case fill the scan lines w/ random data from bottom to top.
/////////////////////////////////////////////////////////////////////////////
CoXKey::CoXKey(unsigned user_key, BITMAPINFO *bmi, LPSTR lpDIBBits)
{
    char    *p_line;
    int     width_in_bytes, line_cnt, i, j, line;
    BOOL    bottom_up;

    this->user_key = user_key;              // save copy of the user's key

    image_data = lpDIBBits;                 // save huge ptr to image data


    // Set up a pointer to the BITMAPINFOHEADER and RBGQUAD array.
    bmiHeader = &bmi->bmiHeader;
    bmiColors = &bmi->bmiColors[0];

    // Set the pointer to the image data.
    this->lpDIBBits = lpDIBBits;

    // Check to see if this is in a format we handle (currently 8 bit only)
    // Need to throw and exception here.
    if (bmiHeader->biBitCount != 8 && bmiHeader->biBitCount != 24)
      return;

    width_in_bytes = (int) WIDTHBYTES(bmiHeader->biWidth * bmiHeader->biBitCount);
```

```
    // Seed the random number generator
    srand(user_key);

    // Image may be top to bottom or bottom to top.
    // We must generate snow accordingly
    if (bmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = bmiHeader->biHeight - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }

    // Generate snow one image scan line at a time.
    for (line_cnt = 0; line_cnt < bmiHeader->biHeight; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        p_line = &image_data[line * (long) width_in_bytes];
        for (i = 0, j = 0; i < bmiHeader->biWidth; i++)
        {
            if (bmiHeader->biBitCount == 24)
            {
                // For 24 bit color case, need r,g,b snow...
                p_line[j++] = (char) rand();
                p_line[j++] = (char) rand();
                p_line[j++] = (char) rand();
            }
            else
            {
                // For test to make grey-scale and color keys match
                // we must call rand 3 times, but only keep same value
                // as the green channel of the rgb version.  This way,
                // if we convert color image to greyscale we can read it.
                rand();
                p_line[i] = (char) rand();  // we make grey snow same as green.
                rand();

            }
        }
        if (bottom_up) line--;
        else line++;
    }
}


void CoXKey::UseNewKey(unsigned newkey)
{
  char    *line;
  int     width_in_bytes, line_cnt, i;

  // Save the new key.
  user_key = newkey;

  width_in_bytes = (int) WIDTHBYTES(bmiHeader->biWidth * bmiHeader->biBitCount);

  // Seed the random number generator
  srand(user_key);

  for (line_cnt = 0; line_cnt < bmiHeader->biHeight; line_cnt++)
  {
      // Set pointer to first byte for this scan line.
      line = &image_data[line_cnt * (long) width_in_bytes];
      for (i = 0; i < bmiHeader->biWidth; i++)
      {
          line[i] = (char) rand();
      }
  }
}
```

COXKEY.H

```
/**********************************************************************\
* FILE: CoXKey.h                                                       *
*                                                                      *
* DESCRIPTION:                                                         *
* The CoXKey (for Coextensive Key) class encapsulates the functions and *
* data structures used to generate a "snowy image" of the same extent  *
* (i.e., x, y dimensions) as the input image.                          *
*                                                                      *
* This header file should be included by any module which creates or   *
* makes use of coXKey objects.                                         *
*                                                                      *
* CREATION DATE:  August 15, 1995                                      *
*                                                                      *
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.       *
\**********************************************************************/

#ifndef COXKEY_H
#define COXKEY_H

//#include "digimark.h"
//#include "params.h"
//#include "RawImage.h"
#include    "stdafx.h"
//#include  "afx.h"


class CoXKey
{

// Public member functions
public:
    // The constructor is passed the user key value and ptrs to the DIB header
    // structures and the data space.   The header is assumed to be filled out
    // correctly, while the data space is allocated but empty.
    // Alternative: pass an HDIB handle, allowing this class to handle locking.
    // FOR NOW, I ALSO ASSUME THE PALETTE HAS BEEN SET UP (its the same as image we are
signing)
    // CoXKey(int user_key, HDIB hDib);
    CoXKey(unsigned user_key, BITMAPINFO *bmi, LPSTR lpDIBBits);


// Private member functions
private:
    // This function may be a useful idea for future, but it needs rework.
    // I'm making it private to assure no one is calling it.
    void UseNewKey(unsigned newkey);


// Private data
private:

    // Copy of the user key value.
    unsigned            user_key;

    // Pointers to the bitmap info header structure, and the palette array.
    BITMAPINFOHEADER    *bmiHeader;     // Points to header structure
    RGBQUAD             *bmiColors;     // Pts to beginning of palette array

    LPSTR               lpDIBBits;      // Pointer to DIB bits
    char                *image_data;    // Pointer to raw image data.

};

#endif  // COXKEY_H
```

DIPAPI.CPP

```
//  dibapi.cpp
//
//  Source file for Device-Independent Bitmap (DIB) API.  Provides
//  the following functions:
//
//  PaintDIB()          - Painting routine for a DIB
//  CreateDIBPalette()  - Creates a palette from a DIB
//  FindDIBBits()       - Returns a pointer to the DIB bits
//  DIBWidth()          - Gets the width of the DIB
//  DIBHeight()         - Gets the height of the DIB
//  PaletteSize()       - Gets the size required to store the DIB's palette
//  DIBNumColors()      - Calculates the number of colors
//                        in the DIB's color table
//  CopyHandle()        - Makes a copy of the given global memory block
//
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#include "stdafx.h"
#include "dibapi.h"
#include <io.h>
#include <errno.h>

/*************************************************************************
 *
 * PaintDIB()
 *
 * Parameters:
 *
```

```
 * HDC hDC           - DC to do output to
 *
 * LPRECT lpDCRect   - rectangle on DC to do output to
 *
 * HDIB hDIB         - handle to global memory with a DIB spec
 *                     in it followed by the DIB bits
 *
 * LPRECT lpDIBRect  - rectangle of DIB to output into lpDCRect
 *
 * CPalette* pPal    - pointer to CPalette containing DIB's palette
 *
 * Return Value:
 *
 * BOOL              - TRUE if DIB was drawn, FALSE otherwise
 *
 * Description:
 *   Painting routine for a DIB.  Calls StretchDIBits() or
 *   SetDIBitsToDevice() to paint the DIB.  The DIB is
 *   output to the specified DC, at the coordinates given
 *   in lpDCRect.  The area of the DIB to be output is
 *   given by lpDIBRect.
 *
 ************************************************************************/

BOOL WINAPI PaintDIB(HDC      hDC,
                     LPRECT   lpDCRect,
                     HDIB     hDIB,
                     LPRECT   lpDIBRect,
                     CPalette* pPal)
{
  LPSTR     lpDIBHdr;             // Pointer to BITMAPINFOHEADER
  LPSTR     lpDIBBits;            // Pointer to DIB bits
  BOOL      bSuccess=FALSE;       // Success/fail flag
  HPALETTE hPal=NULL;             // Our DIB's palette
  HPALETTE hOldPal=NULL;          // Previous palette

  /* Check for valid DIB handle */
  if (hDIB == NULL)
      return FALSE;

  /* Lock down the DIB, and get a pointer to the beginning of the bit
   *  buffer
   */
  lpDIBHdr  = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);
  lpDIBBits = ::FindDIBBits(lpDIBHdr);

  // Get the DIB's palette, then select it into DC
  if (pPal != NULL)
  {
      hPal = (HPALETTE) pPal->m_hObject;

      // Select as background since we have
      // already realized in forground if needed
      hOldPal = ::SelectPalette(hDC, hPal, TRUE);
  }

  /* Make sure to use the stretching mode best for color pictures */
  ::SetStretchBltMode(hDC, COLORONCOLOR);

  /* Determine whether to call StretchDIBits() or SetDIBitsToDevice() */
  if ((RECTWIDTH(lpDCRect)  == RECTWIDTH(lpDIBRect)) &&
     (RECTHEIGHT(lpDCRect)  == RECTHEIGHT(lpDIBRect)))
     bSuccess = ::SetDIBitsToDevice(hDC,                      // hDC
                             lpDCRect->left,                  // DestX
                             lpDCRect->top,                   // DestY
                             RECTWIDTH(lpDCRect),             // nDestWidth
                             RECTHEIGHT(lpDCRect),            // nDestHeight
                             lpDIBRect->left,                 // SrcX
                             (int)DIBHeight(lpDIBHdr) -
                                lpDIBRect->top -
                                RECTHEIGHT(lpDIBRect),        // SrcY
                             0,                               // nStartScan
                             (WORD)DIBHeight(lpDIBHdr),       // nNumScans
                             lpDIBBits,                       // lpBits
                             (LPBITMAPINFO)lpDIBHdr,          // lpBitsInfo
                             DIB_RGB_COLORS);                 // wUsage
   else
    bSuccess = ::StretchDIBits(hDC,                           // hDC
                         lpDCRect->left,                      // DestX
                         lpDCRect->top,                       // DestY
                         RECTWIDTH(lpDCRect),                 // nDestWidth
                         RECTHEIGHT(lpDCRect),                // nDestHeight
                         lpDIBRect->left,                     // SrcX
                         lpDIBRect->top,                      // SrcY
                         RECTWIDTH(lpDIBRect),                // wSrcWidth
                         RECTHEIGHT(lpDIBRect),               // wSrcHeight
                         lpDIBBits,                           // lpBits
                         (LPBITMAPINFO)lpDIBHdr,              // lpBitsInfo
                         DIB_RGB_COLORS,                      // wUsage
                         SRCCOPY);                            // dwROP

      ::GlobalUnlock((HGLOBAL) hDIB);

      /* Reselect old palette */
      if (hOldPal != NULL)
      {
          ::SelectPalette(hDC, hOldPal, TRUE);
      }

      return bSuccess;
}

/*************************************************************************
 *
 * CreateDIBPalette()
 *
 * Parameter:
 *
 * HDIB hDIB         - specifies the DIB
 *
 * Return Value:
 *
 * HPALETTE          - specifies the palette
 *
 * Description:
 *
 * This function creates a palette from a DIB by allocating memory for the
 * logical palette, reading and storing the colors from the DIB's color table
 * into the logical palette, creating a palette from this logical palette,
 * and then returning the palette's handle. This allows the DIB to be
 * displayed using the best possible colors (important for DIBs with 256 or
 * more colors).
 *
 ************************************************************************/


BOOL WINAPI CreateDIBPalette(HDIB hDIB, CPalette* pPal)
{
    LPLOGPALETTE lpPal;       // pointer to a logical palette
    HANDLE hLogPal;           // handle to a logical palette
    HPALETTE hPal = NULL;     // handle to a palette
    int i;                    // loop index
    WORD wNumColors;          // number of colors in color table
    LPSTR lpbi;               // pointer to packed-DIB
    LPBITMAPINFO lpbmi;       // pointer to BITMAPINFO structure (Win3.0)
    LPBITMAPCOREINFO lpbmc;   // pointer to BITMAPCOREINFO structure (old)
    BOOL bWinStyleDIB;        // flag which signifies whether this is a Win3.0 DIB
    BOOL bResult = FALSE;

    /* if handle to DIB is invalid, return FALSE */

    if (hDIB == NULL)
      return FALSE;

   lpbi = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);

   /* get pointer to BITMAPINFO (Win 3.0) */
   lpbmi = (LPBITMAPINFO)lpbi;

   /* get pointer to BITMAPCOREINFO (old 1.x) */
   lpbmc = (LPBITMAPCOREINFO)lpbi;

   /* get the number of colors in the DIB */
   wNumColors = ::DIBNumColors(lpbi);

   if (wNumColors != 0)
   {
        /* allocate memory block for logical palette */
        hLogPal = ::GlobalAlloc(GHND, sizeof(LOGPALETTE)
                                    + sizeof(PALETTEENTRY)
                                    * wNumColors);

        /* if not enough memory, clean up and return NULL */
        if (hLogPal == 0)
        {
            ::GlobalUnlock((HGLOBAL) hDIB);
            return FALSE;
        }

        lpPal = (LPLOGPALETTE) ::GlobalLock((HGLOBAL) hLogPal);

        /* set version and number of palette entries */
        lpPal->palVersion = PALVERSION;
        lpPal->palNumEntries = (WORD)wNumColors;

        /* is this a Win 3.0 DIB? */
        bWinStyleDIB = IS_WIN30_DIB(lpbi);
        for (i = 0; i < (int)wNumColors; i++)
        {
            if (bWinStyleDIB)
            {
```

```
            lpPal->palPalEntry[i].peRed = lpbmi->bmiColors[i].rgbRed;
            lpPal->palPalEntry[i].peGreen = lpbmi->bmiColors[i].rgbGreen;
            lpPal->palPalEntry[i].peBlue = lpbmi->bmiColors[i].rgbBlue;
            lpPal->palPalEntry[i].peFlags = 0;
        }
        else
        {
            lpPal->palPalEntry[i].peRed = lpbmc->bmciColors[i].rgbtRed;
            lpPal->palPalEntry[i].peGreen = lpbmc->bmciColors[i].rgbtGreen;
            lpPal->palPalEntry[i].peBlue = lpbmc->bmciColors[i].rgbtBlue;
            lpPal->palPalEntry[i].peFlags = 0;
        }
    }

    /* create the palette and get handle to it */
    bResult = pPal->CreatePalette(lpPal);
    ::GlobalUnlock((HGLOBAL) hLogPal);
    ::GlobalFree((HGLOBAL) hLogPal);
  }

  ::GlobalUnlock((HGLOBAL) hDIB);

  return bResult;
}

/*************************************************************************
 *
 * FindDIBBits()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * LPSTR            - pointer to the DIB bits
 *
 * Description:
 *
 * This function calculates the address of the DIB's bits and returns a
 * pointer to the DIB bits.
 *
 ************************************************************************/


LPSTR WINAPI FindDIBBits(LPSTR lpbi)
{
  return (lpbi + *(LPDWORD)lpbi + ::PaletteSize(lpbi));
}

/*************************************************************************
 *
 * DIBWidth()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * DWORD            - width of the DIB
 *
 * Description:
 *
 * This function gets the width of the DIB from the BITMAPINFOHEADER
 * width field if it is a Windows 3.0-style DIB or from the BITMAPCOREHEADER
 * width field if it is an other-style DIB.
 *
 ************************************************************************/


DWORD WINAPI DIBWidth(LPSTR lpDIB)
{
  LPBITMAPINFOHEADER lpbmi;  // pointer to a Win 3.0-style DIB
  LPBITMAPCOREHEADER lpbmc;  // pointer to an other-style DIB

  /* point to the header (whether Win 3.0 and old) */

  lpbmi = (LPBITMAPINFOHEADER)lpDIB;
  lpbmc = (LPBITMAPCOREHEADER)lpDIB;

  /* return the DIB width if it is a Win 3.0 DIB */
  if (IS_WIN30_DIB(lpDIB))
      return lpbmi->biWidth;
  else  /* it is an other-style DIB, so return its width */
      return (DWORD)lpbmc->bcWidth;
}

/*************************************************************************
 *
 * DIBHeight()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * DWORD            - height of the DIB
 *
 * Description:
 *
 * This function gets the height of the DIB from the BITMAPINFOHEADER
 * height field if it is a Windows 3.0-style DIB or from the BITMAPCOREHEADER
 * height field if it is an other-style DIB.
 *
 ************************************************************************/


DWORD WINAPI DIBHeight(LPSTR lpDIB)
{
    LPBITMAPINFOHEADER lpbmi;  // pointer to a Win 3.0-style DIB
    LPBITMAPCOREHEADER lpbmc;  // pointer to an other-style DIB

    /* point to the header (whether old or Win 3.0 */

    lpbmi = (LPBITMAPINFOHEADER)lpDIB;
    lpbmc = (LPBITMAPCOREHEADER)lpDIB;

    /* return the DIB height if it is a Win 3.0 DIB */
    if (IS_WIN30_DIB(lpDIB))
        return lpbmi->biHeight;
    else  /* it is an other-style DIB, so return its height */
        return (DWORD)lpbmc->bcHeight;
}

/*************************************************************************
 *
 * PaletteSize()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * WORD             - size of the color palette of the DIB
 *
 * Description:
 *
 * This function gets the size required to store the DIB's palette by
 * multiplying the number of colors by the size of an RGBQUAD (for a
 * Windows 3.0-style DIB) or by the size of an RGBTRIPLE (for an other-
 * style DIB).
 *
 ************************************************************************/


WORD WINAPI PaletteSize(LPSTR lpbi)
{
   /* calculate the size required by the palette */
   if (IS_WIN30_DIB (lpbi))
      return (WORD)(::DIBNumColors(lpbi) * sizeof(RGBQUAD));
   else
      return (WORD)(::DIBNumColors(lpbi) * sizeof(RGBTRIPLE));
}

/*************************************************************************
 *
 * DIBNumColors()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * WORD             - number of colors in the color table
 *
 * Description:
 *
 * This function calculates the number of colors in the DIB's color table
 * by finding the bits per pixel for the DIB (whether Win3.0 or other-style
 * DIB). If bits per pixel is 1: colors=2, if 4: colors=16, if 8: colors=256,
 * if 24, no colors in color table.
 *
```

```
*****************************************************************************/

WORD WINAPI DIBNumColors(LPSTR lpbi)
{
  WORD wBitCount;  // DIB bit count

  /*  If this is a Windows-style DIB, the number of colors in the
   *  color table can be less than the number of bits per pixel
   *  allows for (i.e. lpbi->biClrUsed can be set to some value).
   *  If this is the case, return the appropriate value.
   */

  if (IS_WIN30_DIB(lpbi))
  {
      DWORD dwClrUsed;

      dwClrUsed = ((LPBITMAPINFOHEADER)lpbi)->biClrUsed;
      if (dwClrUsed != 0)
          return (WORD)dwClrUsed;
  }

  /*  Calculate the number of colors in the color table based on
   *  the number of bits per pixel for the DIB.
   */
  if (IS_WIN30_DIB(lpbi))
      wBitCount = ((LPBITMAPINFOHEADER)lpbi)->biBitCount;
  else
      wBitCount = ((LPBITMAPCOREHEADER)lpbi)->bcBitCount;

  /* return number of colors based on bits per pixel */
  switch (wBitCount)
  {
      case 1:
          return 2;

      case 4:
          return 16;

      case 8:
          return 256;

      default:
          return 0;
  }
}


/*****************************************************************************
 *
 * DIBBitCount()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * WORD             - number of bits per pixel
 *
 * Description:
 *
 * Added by Clay Davidson 11/7/95.  Simply returns the number of bits per
 * pixel (i.e., 2, 4, 8, 24), regardless of the state of the color table.
 *****************************************************************************/
WORD WINAPI DIBBitCount(LPSTR lpbi)
{
  WORD wBitCount;

  if (IS_WIN30_DIB(lpbi))
      wBitCount = ((LPBITMAPINFOHEADER)lpbi)->biBitCount;
  else
      wBitCount = ((LPBITMAPCOREHEADER)lpbi)->bcBitCount;

  return wBitCount;
}

//////////////////////////////////////////////////////////////////////////
//// Clipboard support

//---------------------------------------------------------------------
//
// Function:   CopyHandle (from SDK DibView sample clipbrd.c)
//
// Purpose:    Makes a copy of the given global memory block.  Returns
//             a handle to the new memory block (NULL on error).
//
//             Routine stolen verbatim out of ShowDIB.
//
// Parms:      h == Handle to global memory to duplicate.
//
// Returns:    Handle to new global memory block.
//
//---------------------------------------------------------------------

HANDLE WINAPI CopyHandle (HANDLE h)
{
    BYTE    *lpCopy;
    BYTE    *lp;
    HANDLE  hCopy;
    DWORD   dwLen;

    if (h == NULL)
        return NULL;

    dwLen = ::GlobalSize((HGLOBAL) h);

    if ((hCopy = (HANDLE) ::GlobalAlloc (GHND, dwLen)) != NULL)
    {
        lpCopy = (BYTE *) ::GlobalLock((HGLOBAL) hCopy);
        lp     = (BYTE *) ::GlobalLock((HGLOBAL) h);

        while (dwLen--)
            *lpCopy++ = *lp++;

        ::GlobalUnlock((HGLOBAL) hCopy);
        ::GlobalUnlock((HGLOBAL) h);
    }

    return hCopy;
}
```

DIBAPI.H

```
// dibapi.h

// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.


#ifndef _INC_DIBAPI
#define _INC_DIBAPI

/* Handle to a DIB */
DECLARE_HANDLE(HDIB);

/* DIB constants */
#define PALVERSION   0x300

/* DIB Macros*/

#define IS_WIN30_DIB(lpbi)  ((*(LPDWORD)(lpbi)) == sizeof(BITMAPINFOHEADER))
#define RECTWIDTH(lpRect)     ((lpRect)->right - (lpRect)->left)
#define RECTHEIGHT(lpRect)    ((lpRect)->bottom - (lpRect)->top)

// WIDTHBYTES performs DWORD-aligning of DIB scanlines.  The "bits"
// parameter is the bit count for the scanline (biWidth * biBitCount),
// and this macro returns the number of DWORD-aligned bytes needed
// to hold those bits.

#define WIDTHBYTES(bits)    (((bits) + 31) / 32 * 4)

/* Function prototypes */
BOOL      WINAPI  PaintDIB (HDC, LPRECT, HDIB, LPRECT, CPalette* pPal);
BOOL      WINAPI  CreateDIBPalette(HDIB hDIB, CPalette* cPal);
LPSTR     WINAPI  FindDIBBits (LPSTR lpbi);
DWORD     WINAPI  DIBWidth (LPSTR lpDIB);
DWORD     WINAPI  DIBHeight (LPSTR lpDIB);
WORD      WINAPI  PaletteSize (LPSTR lpbi);
WORD      WINAPI  DIBNumColors (LPSTR lpbi);
WORD      WINAPI  DIBBitCount(LPSTR lpbi);
HANDLE    WINAPI  CopyHandle (HANDLE h);

BOOL      WINAPI  SaveDIB (HDIB hDib, CFile& file);
HDIB      WINAPI  ReadDIBFile(CFile& file);

#endif //!_INC_DIBAPI
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <memory.h>

#define MAX_LINEAR_DIMENSION 4096


/* generates ascii lines for t1-n */
/*
main(){
  int i;

  printf("\n\n");
  for(i=0;i<512;i++){
      printf("%d,",irvb(i,9));
      if( !(i%16) )printf("\n");
  }
  printf("\n\n");
  for(i=0;i<1024;i++){
      printf("%d,",irvb(i,10));
      if( !(i%16) )printf("\n");
  }
}
*/



static    int t1[] = { 0, 1 } ;
static    int t2[] = { 0, 2, 1, 3 } ;
static    int t3[] = { 0, 4, 2, 6, 1, 5, 3, 7 } ;
static    int t4[] = { 0, 8, 4,12, 2,10, 6,14, 1, 9, 5,13, 3,11, 7,15 } ;
static    int t5[] = { 0,16, 8,24, 4,20,12,28, 2,18,10,26, 6,22,14,30,
           1,17, 9,25, 5,21,13,29, 3,19,11,27, 7,23,15,31 } ;
static    int t6[] = { 0,32,16,48, 8,40,24,56, 4,36,20,52,12,44,28,60,
           2,34,18,50,10,42,26,58, 6,38,22,54,14,46,30,62,
           1,33,17,49, 9,41,25,57, 5,37,21,53,13,45,29,61,
           3,35,19,51,11,43,27,59, 7,39,23,55,15,47,31,63 } ;
static    int t7[] = {
0,64,32,96,16,80,48,112,8,72,40,104,24,88,56,120,
4,68,36,100,20,84,52,116,12,76,44,108,28,92,60,124,
2,66,34,98,18,82,50,114,10,74,42,106,26,90,58,122,
6,70,38,102,22,86,54,118,14,78,46,110,30,94,62,126,
1,65,33,97,17,81,49,113,9,73,41,105,25,89,57,121,
5,69,37,101,21,85,53,117,13,77,45,109,29,93,61,125,
3,67,35,99,19,83,51,115,11,75,43,107,27,91,59,123,
7,71,39,103,23,87,55,119,15,79,47,111,31,95,63,127 };

static    int t8[] = {
0,128,64,192,32,160,96,224,16,144,80,208,48,176,112,240,
8,136,72,200,40,168,104,232,24,152,88,216,56,184,120,248,
4,132,68,196,36,164,100,228,20,148,84,212,52,180,116,244,
12,140,76,204,44,172,108,236,28,156,92,220,60,188,124,252,
2,130,66,194,34,162,98,226,18,146,82,210,50,178,114,242,
10,138,74,202,42,170,106,234,26,154,90,218,58,186,122,250,
6,134,70,198,38,166,102,230,22,150,86,214,54,182,118,246,
14,142,78,206,46,174,110,238,30,158,94,222,62,190,126,254,
1,129,65,193,33,161,97,225,17,145,81,209,49,177,113,241,
9,137,73,201,41,169,105,233,25,153,89,217,57,185,121,249,
5,133,69,197,37,165,101,229,21,149,85,213,53,181,117,245,
13,141,77,205,45,173,109,237,29,157,93,221,61,189,125,253,
3,131,67,195,35,163,99,227,19,147,83,211,51,179,115,243,
11,139,75,203,43,171,107,235,27,155,91,219,59,187,123,251,
7,135,71,199,39,167,103,231,23,151,87,215,55,183,119,247,
15,143,79,207,47,175,111,239,31,159,95,223,63,191,127,255 };



static    int t9[] = {
0,
256,128,384,64,320,192,448,32,288,160,416,96,352,224,480,16,
272,144,400,80,336,208,464,48,304,176,432,112,368,240,496,8,
264,136,392,72,328,200,456,40,296,168,424,104,360,232,488,24,
280,152,408,88,344,216,472,56,312,184,440,120,376,248,504,4,
260,132,388,68,324,196,452,36,292,164,420,100,356,228,484,20,
276,148,404,84,340,212,468,52,308,180,436,116,372,244,500,12,
268,140,396,76,332,204,460,44,300,172,428,108,364,236,492,28,
284,156,412,92,348,220,476,60,316,188,444,124,380,252,508,2,
258,130,386,66,322,194,450,34,290,162,418,98,354,226,482,18,
274,146,402,82,338,210,466,50,306,178,434,114,370,242,498,10,
266,138,394,74,330,202,458,42,298,170,426,106,362,234,490,26,
282,154,410,90,346,218,474,58,314,186,442,122,378,250,506,6,
262,134,390,70,326,198,454,38,294,166,422,102,358,230,486,22,
278,150,406,86,342,214,470,54,310,182,438,118,374,246,502,14,
270,142,398,78,334,206,462,46,302,174,430,110,366,238,494,30,
286,158,414,94,350,222,478,62,318,190,446,126,382,254,510,1,
257,129,385,65,321,193,449,33,289,161,417,97,353,225,481,17,
273,145,401,81,337,209,465,49,305,177,433,113,369,241,497,9,
265,137,393,73,329,201,457,41,297,169,425,105,361,233,489,25,
281,153,409,89,345,217,473,57,313,185,441,121,377,249,505,5,
261,133,389,69,325,197,453,37,293,165,421,101,357,229,485,21,
277,149,405,85,341,213,469,53,309,181,437,117,373,245,501,13,
269,141,397,77,333,205,461,45,301,173,429,109,365,237,493,29,
285,157,413,93,349,221,477,61,317,189,445,125,381,253,509,3,
259,131,387,67,323,195,451,35,291,163,419,99,355,227,483,19,
275,147,403,83,339,211,467,51,307,179,435,115,371,243,499,11,
267,139,395,75,331,203,459,43,299,171,427,107,363,235,491,27,
283,155,411,91,347,219,475,59,315,187,443,123,379,251,507,7,
263,135,391,71,327,199,455,39,295,167,423,103,359,231,487,23,
279,151,407,87,343,215,471,55,311,183,439,119,375,247,503,15,
271,143,399,79,335,207,463,47,303,175,431,111,367,239,495,31,
287,159,415,95,351,223,479,63,319,191,447,127,383,255,511  };

static  int t10[] = {
0,
512,256,768,128,640,384,896,64,576,320,832,192,704,448,960,32,
544,288,800,160,672,416,928,96,608,352,864,224,736,480,992,16,
528,272,784,144,656,400,912,80,592,336,848,208,720,464,976,48,
560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
536,280,792,152,664,408,920,88,600,344,856,216,728,472,984,56,
568,312,824,184,696,440,952,120,632,376,888,248,760,504,1016,4,
516,260,772,132,644,388,900,68,580,324,836,196,708,452,964,36,
548,292,804,164,676,420,932,100,612,356,868,228,740,484,996,20,
532,276,788,148,660,404,916,84,596,340,852,212,724,468,980,52,
564,308,820,180,692,436,948,116,628,372,884,244,756,500,1012,12,
524,268,780,140,652,396,908,76,588,332,844,204,716,460,972,44,
556,300,812,172,684,428,940,108,620,364,876,236,748,492,1004,28,
540,284,796,156,668,412,924,92,604,348,860,220,732,476,988,60,
572,316,828,188,700,444,956,124,636,380,892,252,764,508,1020,2,
514,258,770,130,642,386,898,66,578,322,834,194,706,450,962,34,
546,290,802,162,674,418,930,98,610,354,866,226,738,482,994,18,
530,274,786,146,658,402,914,82,594,338,850,210,722,466,978,50,
562,306,818,178,690,434,946,114,626,370,882,242,754,498,1010,10,
522,266,778,138,650,394,906,74,586,330,842,202,714,458,970,42,
554,298,810,170,682,426,938,106,618,362,874,234,746,490,1002,26,
538,282,794,154,666,410,922,90,602,346,858,218,730,474,986,58,
570,314,826,186,698,442,954,122,634,378,890,250,762,506,1018,6,
518,262,774,134,646,390,902,70,582,326,838,198,710,454,966,38,
550,294,806,166,678,422,934,102,614,358,870,230,742,486,998,22,
534,278,790,150,662,406,918,86,598,342,854,214,726,470,982,54,
566,310,822,182,694,438,950,118,630,374,886,246,758,502,1014,14,
526,270,782,142,654,398,910,78,590,334,846,206,718,462,974,46,
558,302,814,174,686,430,942,110,622,366,878,238,750,494,1006,30,
542,286,798,158,670,414,926,94,606,350,862,222,734,478,990,62,
574,318,830,190,702,446,958,126,638,382,894,254,766,510,1022,1,
513,257,769,129,641,385,897,65,577,321,833,193,705,449,961,33,
545,289,801,161,673,417,929,97,609,353,865,225,737,481,993,17,
529,273,785,145,657,401,913,81,593,337,849,209,721,465,977,49,
561,305,817,177,689,433,945,113,625,369,881,241,753,497,1009,9,
521,265,777,137,649,393,905,73,585,329,841,201,713,457,969,41,
553,297,809,169,681,425,937,105,617,361,873,233,745,489,1001,25,
537,281,793,153,665,409,921,89,601,345,857,217,729,473,985,57,
569,313,825,185,697,441,953,121,633,377,889,249,761,505,1017,5,
517,261,773,133,645,389,901,69,581,325,837,197,709,453,965,37,
549,293,805,165,677,421,933,101,613,357,869,229,741,485,997,21,
533,277,789,149,661,405,917,85,597,341,853,213,725,469,981,53,
565,309,821,181,693,437,949,117,629,373,885,245,757,501,1013,13,
525,269,781,141,653,397,909,77,589,333,845,205,717,461,973,45,
557,301,813,173,685,429,941,109,621,365,877,237,749,493,1005,29,
541,285,797,157,669,413,925,93,605,349,861,221,733,477,989,61,
573,317,829,189,701,445,957,125,637,381,893,253,765,509,1021,3,
515,259,771,131,643,387,899,67,579,323,835,195,707,451,963,35,
547,291,803,163,675,419,931,99,611,355,867,227,739,483,995,19,
531,275,787,147,659,403,915,83,595,339,851,211,723,467,979,51,
563,307,819,179,691,435,947,115,627,371,883,243,755,499,1011,11,
523,267,779,139,651,395,907,75,587,331,843,203,715,459,971,43,
555,299,811,171,683,427,939,107,619,363,875,235,747,491,1003,27,
539,283,795,155,667,411,923,91,603,347,859,219,731,475,987,59,
571,315,827,187,699,443,955,123,635,379,891,251,763,507,1019,7,
519,263,775,135,647,391,903,71,583,327,839,199,711,455,967,39,
551,295,807,167,679,423,935,103,615,359,871,231,743,487,999,23,
535,279,791,151,663,407,919,87,599,343,855,215,727,471,983,55,
567,311,823,183,695,439,951,119,631,375,887,247,759,503,1015,15,
527,271,783,143,655,399,911,79,591,335,847,207,719,463,975,47,
559,303,815,175,687,431,943,111,623,367,879,239,751,495,1007,31,
543,287,799,159,671,415,927,95,607,351,863,223,735,479,991,63,
575,319,831,191,703,447,959,127,639,383,895,255,767,511,1023  };



/*
 * irvb() is a routine that returns a number with its bits reversed.
```

```
*/
static int irvb(int n ,int b)
{
  register  int   r ;
  register  int   i ;
  register  int   nn ;
  register  int   bb ;

  bb = b ;
  nn = n ;

  switch( bb )
  {
      case 1 : return( t1[nn] ) ;
      case 2 : return( t2[nn] ) ;
      case 3 : return( t3[nn] ) ;
      case 4 : return( t4[nn] ) ;
      case 5 : return( t5[nn] ) ;
      case 6 : return( t6[nn] ) ;
      case 7 : return( t7[nn] ) ;
      case 8 : return( t8[nn] ) ;
      case 9 : return( t9[nn] ) ;
      case 10 : return( t10[nn] ) ;
      default:
      {
          r = 0 ;

          for( i = 0 ; i < bb ; i++ )
          {
              r = r << 1 ;
              r = r | ( nn & 1 ) ;
              nn = nn >> 1 ;
          }

          return( r ) ;
      }
  }
}

/*
 * fft() is a routine that calculates the discrete Fourier transform
 * of two arrays taken to be the real and the imaginary parts of an
 * complex array. It returns the transform in the arrays.
 */
void fft(float *ar,float *ai,int nbits,int inv,float *wr,float *wi,int neww )
// float  *ar ;        /* the real part of the array */
// float  *ai ;        /* the imag part of the array */
//int nbits ;     /* log base 2 of the number of elements in the arrays*/
//int inv ;       /* nonzero to indicate the inverse transform */
//float   *wr ;        /* the real part of an array of coeficients */
//float   *wi ;        /* the imag part of an array of coeficients */
//int neww ;      /* nonzero to indicate the coeficients must be calced*/
{
  register float  *aar ;
  register float  *aai ;
  register float  *pr1 ;
  register float  *pi1 ;
  register float  *pr2 ;
  register float  *pi2 ;
  register float  r1 ;
  register float  i1 ;
  register float  r2 ;
  register float  i2 ;
  int i ;
  register  int   j ;
  int n ;
  float   fn ;
  float   tpin ;
  register  int   n2 ;
  register  int   n1 ;
  int nb ;
  int nblock ;
  register  int   nsep ;
  register  int   nsep2 ;
  int ns ;
  register float  areal ;
  register float  aimag ;
  register float  wreal ;
  register float  wimag ;
  register float  *pwr ;
  register float  *pwi ;
  float   w ;

  aar = ar ;
  aai = ai ;

  n = 1 << nbits ;
  fn = (float) n ;

  if( inv == 0 )
```

```
  {
      for( i = 0 ; i < n ; i++ )
      {
          aar[i] = aar[i] / fn ;
          aai[i] = -aai[i] / fn ;
      }
  }

  if( neww != 0 )
  {
      tpin = (float)6.283186 / fn ;
      n2 = n / 2 ;

      for( nb = 0 ; nb < n2 ; nb++ )
      {
          w = tpin * ( (float) irvb( nb, nbits-1 ) ) ;
          wr[nb] = (float)cos( (double)w ) ;
          wi[nb] = (float)sin( (double)w ) ;
      }
  }

  nblock = 1 ;
  nsep = n ;

  for( ns = 0 ; ns < nbits ; ns++ )
  {
      nsep2 =  nsep;
      nsep = nsep / 2 ;

      pwr = wr;
      pwi = wi;
      for(nb=0; nb < nblock ; nb++, pwr++,pwi++ )
      {
          n1 = nb*nsep2 ;
          n2 = n1+nsep ;
          pr1 = &aar[n1];
          pr2 = &aar[n2];
          pi1 = &aai[n1];
          pi2 = &aai[n2];
          wreal = *pwr;
          wimag = *pwi;
          for(j=0;j<nsep;j++)
          {
              r1 = *pr1; r2 = *pr2; i1 = *pi1;i2 = *pi2;
              areal = wreal * r2 - wimag * i2;
              aimag = wimag * r2 + wreal * i2;
              *(pr2++) = r1 - areal;
              *(pi2++) = i1 - aimag;
              *(pr1++) = r1 + areal;
              *(pi1++) = i1 + aimag;
          }
      }

      nblock = nblock*2 ;
  }

  for( i = 0 ; i < n ; i++ )
  {
      j = irvb( i, nbits ) ;

      if( i < j )
      {
          areal = aar[i] ;
          aimag = aai[i] ;
          aar[i] = aar[j] ;
          aai[i] = aai[j] ;
          aar[j] = areal ;
          aai[j] = aimag ;
      }

      if( inv == 0 ) aai[i] = -aai[i] ;
  }
}

int fft2d(float *ar,float *ai,int nbits,int inv,float *wr,float *wi )
{
    int i ;
    int j ;
    int ij ;
    int ji ;
    int n ;
    float   xr ;
    float   xi ;

    n = 1 << nbits ;

    for( i = 1 ; i < n ; i++ )
    {
        for( j = 0 ; j < i ; j++ )
        {
```

```
        ij = (i<<nbits)+j ;
        ji = (j<<nbits)+i ;
        xr = ar[ij] ;
        xi = ai[ij] ;
        ar[ij] = ar[ji] ;
        ai[ij] = ai[ji] ;
        ar[ji] = xr ;
        ai[ji] = xi ;
      }
  }

  fft( &ar[0], &ai[0], nbits, inv, wr, wi, 1 ) ;

  for( i = 1 ; i < n ; i++ )
  {
     fft( &ar[i<<nbits], &ai[i<<nbits], nbits, inv, wr, wi, 0 ) ;
  }

  for( i = 1 ; i < n ; i++ )
  {
     for( j = 0 ; j < i ; j++ )
     {
        ij = (i<<nbits)+j ;
        ji = (j<<nbits)+i ;
        xr = ar[ij] ;
        xi = ai[ij] ;
        ar[ij] = ar[ji] ;
        ai[ij] = ai[ji] ;
        ar[ji] = xr ;
        ai[ji] = xi ;
     }
  }

  for( i = 0 ; i < n ; i++ )
  {
     fft( &ar[i<<nbits], &ai[i<<nbits], nbits, inv, wr, wi, 0 ) ;
  }

  return(0);
}

void realfft_two_arrays(float *array1,float *array2,int nbits,int inv,float *wr,float *wi,int neww)
{
  register  int    j ;
  register int     n;
  register int     nhalf;
  float templ[MAX_LINEAR_DIMENSION],temp2[MAX_LINEAR_DIMENSION];
  register float *ptempl;
  register float *ptemp2;
  register float *par;
  register float *par1;
  register float *pai;
  register float *pai1;
  register float *ptemp1_1;
  register float *ptemp2_1;

  n = 1 << nbits ;
  nhalf = n/2;

  if(!inv){
     fft(array1,array2,nbits,inv,wr,wi,neww);
     /* sort the results */
     ptemp1 = temp1;
     ptemp2 = temp2;
     par = array1;
     pai = array2;
     *ptemp1 = *(par++);
     *ptemp2 = *(pai++);
     par1 = &array1[n-1];
     pai1 = &array2[n-1];
     ptemp1+=2;
     ptemp2+=2;
     for(j=1;j<nhalf;j++){
         *(ptemp1++) = (float)0.5 * (*par + *par1);
         *(ptemp2++) = (float)0.5 * (*pai + *pai1);
         *(ptemp1++) = (float)0.5 * (*pai - *pai1);
         *(ptemp2++) = (float)0.5 * (-*par + *par1);
         par++;par1--;pai++;pai1--;
     }
     temp1[1] = *par;
     temp2[1] = *pai;
     /* now copy the results back into original arrays */
     memcpy(array1,temp1,n*sizeof(float));
     memcpy(array2,temp2,n*sizeof(float));
  }
  else {
     /* re-sort results */
     ptemp1 = temp1;
     ptemp2 = temp2;
     par = array1;
     pai = array2;
     *(ptemp1++) = *par;
     *(ptemp2++) = *pai;
     pai+=2;
     par+=2;
     ptemp1_1 = &temp1[n-1];
     ptemp2_1 = &temp2[n-1];
     for(j=1;j<(n/2);j++){
         *(ptemp1++) = (*par - *(pai+1) );
         *(ptemp1_1--) = (*par + *(pai+1));
         *(ptemp2++) = (*(par+1) + *pai );
         *(ptemp2_1--) = (-*(par+1) + *pai);
         pai+=2;
         par+=2;
     }
     *ptemp1 = array1[1];
     *ptemp2 = array2[1];
     fft(array1,array2,nbits,inv,wr,wi,neww);
   }
}

/* this routine requires that the input array have two more rows of n appended, into which the
nyquist
row will be placed */
int realfft2d_in_place(float *ar,int nbits,int inv,float *wr,float *wi )
{
     register  int    i ;
     register  int    j ;
     register  int    ij ;
     register  int    ji ;
     register int     n;
     register int     n2;
     register int     nhalf;
     register float   xr ;
     register float   xi ;
     register float   xr1 ;
     register float   xi1 ;
     float temp_r[MAX_LINEAR_DIMENSION],temp_i[MAX_LINEAR_DIMENSION];
     register float *ptemp_r;
     register float *ptemp_i;
     register float *par;
     register float *par1;
     register float *pai;
     register float *pai1;
     register float *ptemp_r1;
     register float *ptemp_i1;

     n = 1 << nbits ;
     n2 = n*2;
     nhalf = n/2;

if( !inv){

     /* pre-transpose */
     for( i = 1 ; i < n ; i++ )
     {
         for( j = 0 ; j < i ; j++ )
         {
             ij = (i<<nbits)+j ;
             ji = (j<<nbits)+i ;
             xr = ar[ij] ;
             ar[ij] = ar[ji] ;
             ar[ji] = xr ;
         }
     }

     for( i = 0 ; i < nhalf ; i++ )
     {
         if(i==0)fft( &ar[0], &ar[n], nbits, inv, wr, wi, 1 ) ;
         else fft( &ar[n2*i], &ar[n2*i+n], nbits, inv, wr, wi, 0 ) ;

         /* sort and pack results */
         ptemp_r = temp_r;
         ptemp_i = &temp_i[2];
         par = &ar[n2*i];
         par1 = &ar[n2*i+n];
         *(ptemp_r++) = *(par++);
         *(ptemp_r++) = *(par1--);

         pai = &ar[1+n2*i+n];
         pai1 = &ar[n2*i+n2-1];
         for(j=1;j<nhalf;j++){
             *(ptemp_r++) = (float)0.5 * (*par + *par1);
             *(ptemp_r++) = (float)0.5 * (*pai + *pai1);
             *(ptemp_i++) = (float)0.5 * (*pai - *pai1);
             *(ptemp_i++) = (float)0.5 * (-*par + *par1);
             par++;par1--;pai++;pai1--;
         }
         temp_i[0] = *par;
```

```
        temp_i[1] = *pai;

        /* now copy the results back into original arrays */
        memcpy(&ar[n2*i],temp_r,n*sizeof(float));
        memcpy(&ar[n2*i+n],temp_i,n*sizeof(float));
    }

    /* transpose */
    for( i = 2 ; i < n ; i+=2 ) {
        for( j = 0 ; j < i ; j+=2 ) {
            ij = (i<<nbits)+j ;
            ji = (j<<nbits)+i ;
            xr = ar[ij] ;
            xi = ar[ij+n] ;
            xr1 = ar[ij+1] ;
            xi1 = ar[ij+1+n] ;
            ar[ij] = ar[ji] ;
            ar[ij+n] = ar[ji+n] ;
            ar[ij+1] = ar[ji+1] ;
            ar[ij+n+1] = ar[ji+n+1] ;
            ar[ji] = xr ;
            ar[ji+n] = xi ;
            ar[ji+1] = xr1 ;
            ar[ji+n+1] = xi1 ;
        }
    }

    /* place nyquist row into n*n row, and zero out their imaginary rows */
    memcpy(&ar[n*n],&ar[n],n*sizeof(float));
    memset(&ar[n],0,n*sizeof(float));
    memset(&ar[n*n+n],0,n*sizeof(float));

    for( i = 0 ; i < nhalf+1 ; i++ ) fft( &ar[n2*i], &ar[n2*i+n], nbits, inv, wr, wi, 0 ) ;

    /* finally, shift the arrays in order to simplify external processing */
    for(i=0;i<n+2;i++){
        memcpy(temp_r,&ar[i*n],nhalf*sizeof(float) );
        memcpy(&ar[i*n],&ar[nhalf+i*n],nhalf*sizeof(float) );
        memcpy(&ar[nhalf+i*n],temp_r,nhalf*sizeof(float) );
    }
}
else {
    /* undo format */
    for(i=0;i<(n+2);i++){
        memcpy(temp_r,&ar[i*n],(n/2)*sizeof(float) );
        memcpy(&ar[i*n],&ar[n/2+i*n],(n/2)*sizeof(float) );
        memcpy(&ar[n/2+i*n],temp_r,(n/2)*sizeof(float) );
    }

    fft( &ar[0], &ar[n], nbits, inv, wr, wi, 1 ) ;
    for( i = 1 ; i < (1+n/2) ; i++ ) fft( &ar[(2*i)*n], &ar[(2*i+1)*n], nbits, inv, wr, wi, 0 ) ;

    memcpy(&ar[n],&ar[n*n],n*sizeof(float));

    /* transpose */
    for( i = 2 ; i < n ; i+=2 ) {
        for( j = 0 ; j < i ; j+=2 ) {
            ij = (i<<nbits)+j ;
            ji = (j<<nbits)+i ;
            xr = ar[ij] ;
            xi = ar[ij+n] ;
            xr1 = ar[ij+1] ;
            xi1 = ar[ij+1+n] ;
            ar[ij] = ar[ji] ;
            ar[ij+n] = ar[ji+n] ;
            ar[ij+1] = ar[ji+1] ;
            ar[ij+n+1] = ar[ji+n+1] ;
            ar[ji] = xr ;
            ar[ji+n] = xi ;
            ar[ji+1] = xr1 ;
            ar[ji+n+1] = xi1 ;
        }
    }

    for( i = 0 ; i < (n/2) ; i++ )
    {
        /* re-sort results */
        ptemp_r = temp_r;
        ptemp_i = temp_i;
        par = &ar[(2*i)*n];
        *(ptemp_r++) = *(par++);
        *(ptemp_i++) = *(par++);

        pai = &ar[2+(2*i+1)*n];
        ptemp_r1 = &temp_r[n-1];
        ptemp_i1 = &temp_i[n-1];
        for(j=1;j<(n/2);j++){
            *(ptemp_r++) = (*par - *(pai+1) );
            *(ptemp_r1--) = (*par + *(pai+1));
            *(ptemp_i++) = (*(par+1) + *pai );
            *(ptemp_i1--) = (*(par+1) - *pai);
            pai+=2;
            par+=2;
        }
        *ptemp_r = ar[(2*i+1)*n];
        *ptemp_i = ar[(2*i+1)*n + 1];

        /* now copy the results back into original arrays */
        memcpy(&ar[(2*i)*n],temp_r,n*sizeof(float));
        memcpy(&ar[(1+2*i)*n],temp_i,n*sizeof(float));

        fft( &ar[(2*i)*n], &ar[(2*i+1)*n], nbits, inv, wr, wi, 0 ) ;
    }

    /* post transpose */
    for( i = 1 ; i < n ; i++ )
    {
        for( j = 0 ; j < i ; j++ )
        {
            ij = (i<<nbits)+j ;
            ji = (j<<nbits)+i ;
            xr = ar[ij] ;
            ar[ij] = ar[ji] ;
            ar[ji] = xr ;
        }
    }
}
    return(0);
}
```

FFT.H

```
/*********************************************************************\
* FILE: Fft.h
*
* DESCRIPTION:
* Include file for Geoff's FFT routines.  Callers of the FFT functions
* should include this header file to pick up the function prototypes.
*
* Copyright (C) Digimarc Corporation, 1996, all rights reserved.
\*********************************************************************/
void fft(float  *ar,        /* the real part of the array */
         float  *ai,        /* the imag part of the array */
         int    nbits,      /* log base 2 of the number of elements in the arrays*/
         int    inv,        /* nonzero to indicate the inverse transform */
         float  *wr,        /* the real part of an array of coeficients */
         float  *wi,        /* the imag part of an array of coeficients */
         int    neww);      /* nonzero to indicate the coeficients must be calced*/

int fft2d(float *ar,float *ai,int nbits,int inv,float *wr,float *wi );

void realfft_two_arrays(float *array1, float *array2,
                        int nbits, int inv, float *wr, float *wi, int noww);

int realfft2d_in_place(float *ar,int nbits,int inv,float *wr,float *wi );
```

IMAG BT.CPP

```
//////////////////////////////////////////////////////////////////////
// File: Image.cpp
//
// Contains the implementation for the Image class.  Image objects
// are used to contain the image data, and provide a more convenient
// set of services related to accessing the image data as well as
// attribute variables describing the image.
//////////////////////////////////////////////////////////////////////
#include "Image.h"
#include "dibapi.h"
#include "stdafx.h"

//////////////////////////////////////////////////////////////////////
// Image(HDIB hDIB)
//
// Constructor which creates an Image object, given a handle to
// a DIB which is already in memory.
//////////////////////////////////////////////////////////////////////
Image::Image(HDIB hDIB)
{
    BITMAPINFO  *bmi_info;

    m_hpPackedData = NULL;
    m_fileOK = TRUE;            // its already been opened.
```

```
    m_hDIB = hDIB;

    m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

      // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
      // WE KEEP THE DIB DATA LOCKED IN MEMORY.  FOR THIS IMPLEMENTATION,
      // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RBGQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];

    // Set the pointer to the image data.
    m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);

    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
    m_YDim = m_lpBmiHeader->biHeight;
    m_Compression = m_lpBmiHeader->biCompression;

    m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}

//////////////////////////////////////////////////////////////////////////
// Image(HDIB hDIB)
//
// Constructor which creates an Image object, given the name of a DIB
// or BMP file.
//////////////////////////////////////////////////////////////////////////
Image::Image(CString filename)
{
    CFile           file;
    CFileException  fe;
    BITMAPINFO   *bmi_info;

    m_hpPackedData = NULL;


    if (!file.Open(filename, CFile::modeRead | CFile::shareDenyWrite, &fe))
    {
        CString msg("Error reading image file: ");
        msg += filename;
        MessageBox(NULL, msg, NULL, MB_ICONINFORMATION | MB_OK);
        m_fileOK = FALSE;
    }
    else
        m_fileOK = TRUE;


    // Try to read the DIB file, catch any exceptions.
    TRY
    {
        m_hDIB = ::ReadDIBFile(file);
    }
    CATCH(CFileException, eLoad)
    {
        file.Abort;
        MessageBox(NULL, "Error reading the image file", NULL,
                   MB_ICONINFORMATION | MB_OK);
        m_hDIB = NULL;
        m_fileOK = FALSE;
    }
    END_CATCH


    m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

      // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
      // WE KEEP THE DIB DATA LOCKED IN MEMORY.  FOR THIS IMPLEMENTATION,
      // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RBGQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];

    // Set the pointer to the image data.
    m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);

    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
    m_YDim = m_lpBmiHeader->biHeight;
    m_Compression = m_lpBmiHeader->biCompression;

    m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}

//////////////////////////////////////////////////////////////////////////
// ~Image()
//
// The destructor for the Image class of objects.
//////////////////////////////////////////////////////////////////////////
Image::~Image(void)
{
    ::GlobalUnlock( (HGLOBAL) m_hDIB);

    if (m_hpPackedData != NULL)
    {
        ::GlobalUnlock( (HGLOBAL) m_hPackedData);
        ::GlobalFree( (HGLOBAL) m_hPackedData);
    }

}

//////////////////////////////////////////////////////////////////////////
// MakePackedData()
//
// This function copies the DIB image data into a packed format.  This
// is important for two reasons: 1) the DIB formatted data is arranged
// so that each scan line starts on a long word boundary, so there may
// be up to 3 unused bytes at the end of each scan line in the case of
// 8 bit data.  This arrangement is inconvenient when passing the image
// data to the core algorithms.  Also, 2), if a palette is being used
// (this is the case for all but 24 bit image data), this routine looks
// up the actual image values using the palette and places these values
// in the packed data array.  The member variable m_hPackedData is the
// handle to the packed data.
//
// WARNING: CURRENT IMPLEMENTATION ASSUMES 8 BIT GRAY-SCALE IMAGE DATA
//////////////////////////////////////////////////////////////////////////
void Image::MakePackedData(void)
{
    unsigned char   *hpLine;
    unsigned char   *hpData;
    int             line_cnt, line, i;
    BOOLEAN         bottom_up;

    // Create space and get handle for the packed data of the image.
    m_hPackedData = ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT,
                                  m_XDim * (long) m_YDim);
    if (m_hPackedData == 0)
        AfxThrowMemoryException();

    // Lock the packed data global memory (leave locked until destructor).
    m_hpPackedData = (unsigned char *)::GlobalLock( (HGLOBAL) m_hPackedData);


    hpData = m_hpPackedData;

    // Image may be top to bottom or bottom to top.
    if (m_lpBmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = m_YDim - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }

    // TEST CODE
    // For Geoff: don't let it correct for bottom_up
    bottom_up = FALSE;
    line = 0;

    // Now go through each line and create the packed array.
    for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        hpLine = &m_hpDIBBits[line * (long) m_WidthInBytes];
        for (i = 0; i < m_XDim; i++)
        {
            if (m_BitsPerPixel == 24)
                *hpData++ = hpLine[i];
            else
            {
                // For 8 bit (and any other non 24 bit data) we
                // take the image data to be indices into the color
                // table.  We look up the actual value.  Note we
                // assume grey-scale (i.e., r,g,b triples are all equal -
                // we read the green.
                *hpData++ = m_lpBmiColors[hpLine[i]].rgbGreen;
            }
        }
        if (bottom_up) line--;
        else line++;
    }
}
```

```
///////////////////////////////////////////////////////////////////////
// UnpackData()
//
// This function moves the contents of the packed data array back into
// the DIB data space.  This would be used, for example, after one the
// core algorithms have been used to sign the data in the packed array,
// and we want to update the DIB to reflect the changes.  Note that this
// requires that we create our own palette, since otherwise we don't know
// that the new data values have corresponding entries in the palette.
//
// WARNING: CURRENT IMPLEMENTATION ASSUMES 8 BIT GRAY-SCALE IMAGE DATA
///////////////////////////////////////////////////////////////////////
void Image::UnpackData(void)
{
  unsigned char   *hpLine;
  unsigned char   *hpData;
  int             line_cnt, line, i;
  BOOLEAN         bottom_up;


  // Image may be top to bottom or bottom to top.
  if (m_lpBmiHeader->biHeight > 0)
  {
      bottom_up = TRUE;
      line = m_YDim - 1;
  }
  else
  {
      bottom_up = FALSE;
      line = 0;
  }

  // TEST CODE
  // For Geoff: don't let it correct for bottom_up
  bottom_up = FALSE;
  line = 0;

  hpData = m_hpPackedData;
  for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
  {
      // Set pointer to first byte for this scan line.
      hpLine = &m_hpDIBBits[line * (long)m_WidthInBytes];
      for (i = 0; i < m_XDim; i++)
      {
          hpLine[i] = *hpData++;
      }
      if (bottom_up) line--;
      else line++;
  }

  // Next, we force the palette to be our standard 8 bit grey-scale
  // palette.
  if (m_BitsPerPixel == 8)
  {
      // Set ptr to beginning of palette
      LPRGBQUAD   pal = m_lpBmiColors;

      for (i = 0; i < 256; i++)
      {
          pal[i].rgbBlue = pal[i].rgbGreen = pal[i].rgbRed = i;
      }
  }
  else
  {
      MessageBox(NULL, "Can only unpack 8 bit image data", NULL,
                 MB_ICONEXCLAMATION | MB_OK);
  }

}
```

IMAGE.CPP

```
///////////////////////////////////////////////////////////////////////
// File: Image.cpp
//
// Contains the implementation for the Image class.  Image objects
// are used to contain the image data, and provide a more convenient
// set of services related to accessing the image data as well as
// attribute variables describing the image.
///////////////////////////////////////////////////////////////////////
#include "Image.h"
#include "dibapi.h"
#include "stdafx.h"

///////////////////////////////////////////////////////////////////////
// Image(HDIB hDIB)
//
// Constructor which creates an Image object, given a handle to
// a DIB which is already in memory.
///////////////////////////////////////////////////////////////////////
Image::Image(HDIB hDIB)
{
    BITMAPINFO  *bmi_info;

    m_hpPackedData = NULL;
    m_fileOK = TRUE;            // its already been opened.

    m_hDIB = hDIB;

    m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

    // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
    // WE KEEP THE DIB DATA LOCKED IN MEMORY.  FOR THIS IMPLEMENTATION,
    // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RBGQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];// will be null for 24 bit

    // Set the pointer to the image data.
    m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);

    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
    m_YDim = m_lpBmiHeader->biHeight;
    m_Compression = m_lpBmiHeader->biCompression;

    m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}

///////////////////////////////////////////////////////////////////////
// Image(HDIB hDIB)
//
// Constructor which creates an Image object, given the name of a DIB
// or BMP file.
///////////////////////////////////////////////////////////////////////
Image::Image(CString filename)
{
    CFile           file;
    CFileException  fe;
    BITMAPINFO   *bmi_info;

    m_hpPackedData = NULL;


    if (!file.Open(filename, CFile::modeRead | CFile::shareDenyWrite, &fe))
    {
        CString msg("Error reading image file: ");
        msg += filename;
        MessageBox(NULL, msg, NULL, MB_ICONINFORMATION | MB_OK);
        m_fileOK = FALSE;
    }
    else
        m_fileOK = TRUE;


    // Try to read the DIB file, catch any exceptions.
    TRY
    {
        m_hDIB = ::ReadDIBFile(file);
    }
    CATCH(CFileException, eLoad)
    {
        file.Abort;
        MessageBox(NULL, "Error reading the image file", NULL,
                   MB_ICONINFORMATION | MB_OK);
        m_hDIB = NULL;
        m_fileOK = FALSE;
    }
    END_CATCH


    m_lpDIB = (LPSTR) ::GlobalLock( (HGLOBAL) m_hDIB);

    // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
    // WE KEEP THE DIB DATA LOCKED IN MEMORY.  FOR THIS IMPLEMENTATION,
    // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.

    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RBGQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];

    // Set the pointer to the image data.
    m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);

    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
```

```
    m_YDim = m_lpBmiHeader->biHeight;
    m_Compression = m_lpBmiHeader->biCompression;

    m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}

///////////////////////////////////////////////////////////////////////////
// ~Image()
//
// The destructor for the Image class of objects.
///////////////////////////////////////////////////////////////////////////
Image::~Image(void)
{
    ::GlobalUnlock( (HGLOBAL) m_hDIB);

    if (m_hpPackedData != NULL)
    {
        ::GlobalUnlock( (HGLOBAL) m_hPackedData);
        ::GlobalFree( (HGLOBAL) m_hPackedData);
    }

}

///////////////////////////////////////////////////////////////////////////
// MakePackedData()
//
// This function copies the DIB image data into a packed format.  This
// is important for two reasons: 1) the DIB formatted data is arranged
// so that each scan line starts on a long word boundary, so there may
// be up to 3 unused bytes at the end of each scan line in the case of
// 8 bit data.  This arrangement is inconvenient when passing the image
// data to the core algorithms.  Also, 2), if a palette is being used
// (this is the case for all but 24 bit image data), this routine looks
// up the actual image values using the palette and places these values
// in the packed data array.  The member variable m_hPackedData is the
// handle to the packed data.
//
// The force_to_1_chan argument is an optional boolean.  It defaults
// to FALSE (see function prototype in Image.h).  If set to TRUE,
// only 1 chanel of packed data is created, even if the image is 3
// channels.  This is useful when creating snowy images from RGB
// images, since we currently always want 1 channel snowy images.
///////////////////////////////////////////////////////////////////////////
void Image::MakePackedData(BOOLEAN force_to_1_chan)
{
    unsigned char   *hpLine;
    unsigned char   *hpData;
    int             line_cnt, line, i, j;
    long            size;
    BOOLEAN         bottom_up;

    // Create space and get handle for the packed data of the image.
    size = m_XDim * m_YDim;
    // For 24 bit true color, we will pack R,G,B values, so triple the size.
    if (m_BitsPerPixel == 24 && force_to_1_chan == FALSE)
        size *= 3;
    m_hPackedData = ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, size);
    if (m_hPackedData == 0)
        AfxThrowMemoryException();

    // Lock the packed data global memory (leave locked until destructor).
    m_hpPackedData = (unsigned char *)::GlobalLock( (HGLOBAL) m_hPackedData);


    hpData = m_hpPackedData;

    // Image may be top to bottom or bottom to top.
    if (m_lpBmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = m_YDim - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }

    // TEST CODE
    // For Geoff: don't let it correct for bottom_up
    // bottom_up = FALSE;
    // line = 0;

    // Now go through each line and create the packed array.
    for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        hpLine = &m_hpDIBBits[line * (long) m_WidthInBytes];
        for (i = 0, j = 0; i < m_XDim; i++)
        {
            if (m_BitsPerPixel == 24)
            {
                if (!force_to_1_chan)
                {
                    *hpData++ = hpLine[j+2];// red
                    *hpData++ = hpLine[j+1];// green
                    *hpData++ = hpLine[j+0];// blue
                }
                else
                {
                    *hpData++ = hpLine[j+1];// take just green to convert
                }                           // to 1 channel data.
                j += 3;
            }
            else
            {
                // For 8 bit (and any other non 24 bit data) we
                // take the image data to be indices into the color
                // table.  We look up the actual value.  Note we
                // assume grey-scale (i.e., r,g,b triples are all equal -
                // we read the green.
                *hpData++ = m_lpBmiColors[hpLine[i]].rgbGreen;
            }
        }
        if (bottom_up) line--;
        else line++;
    }
}

///////////////////////////////////////////////////////////////////////////
// UnpackData()
//
// This function moves the contents of the packed data array back into
// the DIB data space.  This would be used, for example, after one the
// core algorithms have been used to sign the data in the packed array,
// and we want to update the DIB to reflect the changes.  Note that this
// requires that we create our own palette, since otherwise we don't know
// that the new data values have corresponding entries in the palette.
//
// WARNING: CURRENT IMPLEMENTATION ASSUMES 8 BIT GRAY-SCALE IMAGE DATA
//          OR 24 BIT COLOR IMAGE DATA
///////////////////////////////////////////////////////////////////////////
void Image::UnpackData(void)
{
    unsigned char   *hpLine;
    unsigned char   *hpData;
    int             line_cnt, line, i, j;
    BOOLEAN         bottom_up;


    // Image may be top to bottom or bottom to top.
    if (m_lpBmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = m_YDim - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }

    // TEST CODE
    // For Geoff: don't let it correct for bottom_up
    // bottom_up = FALSE;
    // line = 0;

    hpData = m_hpPackedData;
    for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        hpLine = &m_hpDIBBits[line * (long) m_WidthInBytes];
        for (i = 0, j = 0; i < m_XDim; i++)
        {
            if (m_BitsPerPixel == 24)
            {
                hpLine[j+2] = *hpData++;    // red
                hpLine[j+1] = *hpData++;    // green
                hpLine[j] = *hpData++;      // blue
                j += 3;
            }
            else
                hpLine[i] = *hpData++;
        }
        if (bottom_up) line--;
        else line++;
    }

    // Next, we force the palette to be our standard 8 bit grey-scale
    // palette.
```

```
    if (m_BitsPerPixel == 8)
    {
        // Set ptr to beginning of palette
        LPRGBQUAD   pal = m_lpBmiColors;

        for (i = 0; i < 256; i++)
        {
            pal[i].rgbBlue = pal[i].rgbGreen = pal[i].rgbRed = i;
        }
    }
    else if (m_BitsPerPixel == 24)
    {
        // Don't do any palette work for 24 bit color: there is no palette.
    }
    else
    {
        MessageBox(NULL, "Can only unpack 8 and 24 bit image data", NULL,
                   MB_ICONEXCLAMATION | MB_OK);
    }

}
```

IMAGE.H

```
/*****************************************************************\
* FILE: Image.h                                                     *
*                                                                   *
* DESCRIPTION:                                                      *
* The Image class is used to read .BMP and .DIB image files, and    *
* manage an internal representation of them in memory.  The goal is *
* to provide a set of service which insulate the caller from having to *
* deal with the specifics of the DIB format.  Also, the approach tends *
* to isolate platform specific and file format specific details to this *
* class.  For example, adding support for a different type of file *
* format would affect this class, but not the callers.             *
*                                                                   *
* This header file should be included by any module which creates or *
* makes use of Image objects.                                       *
*                                                                   *
* CREATION DATE:  September 5, 1995                                 *
*                                                                   *
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.    *
\*****************************************************************/
#ifndef IMAGE_H
#define IMAGE_H

#include  "stdafx.h"
#include  "dibapi.h"


class Image
{
public:
  // Constructors...
  Image(HDIB hDIB);                     // Takes a handle to a loaded DIB
  Image::Image(CString filename);       // Takes a filename
  ~Image(void);
//     void Image::MakePackedData(void);
  void Image::MakePackedData(BOOLEAN force_to_1_chan = FALSE);
  void Image::UnpackData();

  // Accessors:
  HDIB              GetHDIB(void) {return m_hDIB;}
  LPSTR             GetLpDIB(void) {return m_lpDIB;}
  BITMAPINFOHEADER *GetBmiHdr(void) {return m_lpBmiHeader;}
  RGBQUAD          *GetPalette(void) {return m_lpBmiColors;}
  unsigned char    *GetDIBData(void) {return m_hpDIBBits;}
  unsigned char    *GetPackedData(void) {return m_hpPackedData;}
  int               GetBitsPerPixel(void) {return m_BitsPerPixel;}
  WORD              GetSizofPalette(void) {return ::PaletteSize(m_lpDIB);}
  WORD              GetSizeofHeader(void) {return sizeof(BITMAPINFOHEADER) +
                                           ::PaletteSize(m_lpDIB);}
  WORD              GetNumColors(void) {return ::DIBNumColors(m_lpDIB);}
  LONG              GetXDim(void) {return m_XDim;}
  LONG              GetYDim(void) {return m_YDim;}

  BOOL              GetFileOK(void) {return m_fileOK;}

// Private member functions

// Private data
private:

  // Handle to the DIB.
  HDIB                m_hDIB;

  LPSTR               m_lpDIB;          // Pointer to top of DIB, locked in memory

  // Pointers to the bitmap info header structure, and the palette array.
  LPBITMAPINFOHEADER  m_lpBmiHeader;      // Points to header structure
  RGBQUAD         FAR* m_lpBmiColors;     // Pts to beginning of palette array

  unsigned char       *m_hpDIBBits;       // Pointer to DIB bits
  HANDLE              m_hPackedData;      // Handle for the packed data space
  unsigned char       *m_hpPackedData;    // Pointer to Packed copy of data.

  LONG                m_XDim;             // X dimension of image
  LONG                m_YDim;             // Y dimension of image (number of lines)
  int                 m_BitsPerPixel;
  LONG                m_WidthInBytes;     // No. of bytes used in each line of DIB
  DWORD               m_Compression;

  BOOL                m_fileOK;
};

#endif  // IMAGE_H
```

MAINFRM.CPP

```
// mainfrm.cpp : implementation of the CMainFrame class
//


#include "stdafx.h"
#include "signer.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
    // same order as in the bitmap 'toolbar.bmp'
    ID_FILE_NEW,
    ID_FILE_OPEN,
    ID_FILE_SAVE_AS,
        ID_SEPARATOR,
    ID_EDIT_CUT,
    ID_EDIT_COPY,
    ID_EDIT_PASTE,
        ID_SEPARATOR,
    ID_FILE_PRINT,
    ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
  if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;

  if (!m_wndToolBar.Create(this) ||
      !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
      !m_wndToolBar.SetButtons(buttons,
        sizeof(buttons)/sizeof(UINT)))
  {
      TRACE("Failed to create toolbar\n");
      return -1;      // fail to create
  }

  if (!m_wndStatusBar.Create(this) ||
      !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
  {
      TRACE("Failed to create status bar\n");
      return -1;      // fail to create
  }

  return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame commands


void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
  CMDIFrameWnd::OnPaletteChanged(pFocusWnd);

  // always realize the palette for the active view
  CMDIChildWnd* pMDIChildWnd = MDIGetActive();
  if (pMDIChildWnd == NULL)
      return; // no active MDI child frame
  CView* pView = pMDIChildWnd->GetActiveView();
  ASSERT(pView != NULL);

  // notify all child windows that the palette has changed
  SendMessageToDescendants(WM_DOREALIZE, (WPARAM)pView->m_hWnd);
}


BOOL CMainFrame::OnQueryNewPalette()
{
  // always realize the palette for the active view
  CMDIChildWnd* pMDIChildWnd = MDIGetActive();
  if (pMDIChildWnd == NULL)
      return FALSE; // no active MDI child frame (no new palette)
  CView* pView = pMDIChildWnd->GetActiveView();
  ASSERT(pView != NULL);

  // just notify the target view
  pView->SendMessage(WM_DOREALIZE, (WPARAM)pView->m_hWnd);
  return TRUE;
}
```

MAINFRM.H

```
// mainfrm.h : interface of the CMainFrame class
//
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#ifndef __APXEXT_H__
#include <afxext.h>         // for access to CToolBar and CStatusBar
#endif

class CMainFrame : public CMDIFrameWnd
{
  DECLARE_DYNAMIC(CMainFrame)
public:
  CMainFrame();

// Implementation
public:
  virtual ~CMainFrame();

    // Need public access to the CMDIFrameWnd::OnWindowNew() function,
    // in order to programatically create new windows and views.
    void MyOnWindowNew(void) {OnWindowNew();}

protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    afx_msg BOOL OnQueryNewPalette();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

MYCHILDW.CPP

```
// mychildw.cpp : implementation file
//
// This class was created in order to over-ride the
// default behavior of the CMDIChildWnd::PreCreateWindow()
// member function, allowing my view class to create
// a customized child window title.

#include "stdafx.h"
#include "signer.h"
#include "mychildw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMyChildWnd

IMPLEMENT_DYNCREATE(CMyChildWnd, CMDIChildWnd)

CMyChildWnd::CMyChildWnd()
{
}

CMyChildWnd::~CMyChildWnd()
{
}


BEGIN_MESSAGE_MAP(CMyChildWnd, CMDIChildWnd)
    //{{AFX_MSG_MAP(CMyChildWnd)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()


BOOL CMyChildWnd::PreCreateWindow(CREATESTRUCT &cs)
{
    // Do default processing
    if (CMDIChildWnd::PreCreateWindow(cs) == 0)
        return FALSE;
    else
    {
        cs.style &= ~(LONG) FWS_ADDTOTITLE;
        return TRUE;
    }
}

/////////////////////////////////////////////////////////////////////////////
// CMyChildWnd message handlers
```

MYCHILDW.H

```
// mychildw.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CMyChildWnd frame

class CMyChildWnd : public CMDIChildWnd
{
```

```
  DECLARE_DYNCREATE(CMyChildWnd)
protected:
  CMyChildWnd();          // protected constructor used by dynamic creation

// Attributes
public:

// Operations
public:

// Implementation
protected:
  virtual ~CMyChildWnd();

  virtual BOOL PreCreateWindow(CREATESTRUCT &cs);

  // Generated message map functions
  //{{AFX_MSG(CMyChildWnd)
      // NOTE - the ClassWizard will add and remove member functions here.
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

MYFILE.CPP

```
//  myfile.cpp
//
//  Source file for Device-Independent Bitmap (DIB) API.  Provides
//  the following functions:
//
//  SaveDIB()           - Saves the specified dib in a file
//  ReadDIBFile()       - Loads a DIB from a file
//
//
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#include "stdafx.h"
#include <math.h>
#include <io.h>
#include <direct.h>
#include "dibapi.h"

/*
 * Dib Header Marker - used in writing DIBs to files
 */
#define DIB_HEADER_MARKER   ((WORD) ('M' << 8) | 'B')


/*********************************************************************
 *
 * SaveDIB()
 *
 * Saves the specified DIB into the specified CFile.  The CFile
 * is opened and closed by the caller.
 *
 * Parameters:
 *
 * HDIB hDib - Handle to the dib to save
 *
 * CFile& file - open CFile used to save DIB
 *
 * Return value: TRUE if successful, else FALSE or CFileException
 *
 *********************************************************************/


BOOL WINAPI SaveDIB(HDIB hDib, CFile& file)
{
  BITMAPFILEHEADER bmfHdr; // Header for Bitmap file
  LPBITMAPINFOHEADER lpBI;   // Pointer to DIB info structure
  DWORD dwDIBSize;

  if (hDib == NULL)
      return FALSE;

  /*
   * Get a pointer to the DIB memory, the first of which contains
   * a BITMAPINFO structure
   */
  lpBI = (LPBITMAPINFOHEADER) ::GlobalLock((HGLOBAL) hDib);
  if (lpBI == NULL)
      return FALSE;

  if (!IS_WIN30_DIB(lpBI))
  {
      ::GlobalUnlock((HGLOBAL) hDib);
      return FALSE;       // It's an other-style DIB (save not supported)
  }

  /*
   * Fill in the fields of the file header
   */

  /* Fill in file type (first 2 bytes must be "BM" for a bitmap) */
  bmfHdr.bfType = DIB_HEADER_MARKER;  // "BM"

  // Calculating the size of the DIB is a bit tricky (if we want to
  // do it right).  The easiest way to do this is to call GlobalSize()
  // on our global handle, but since the size of our global memory may have
  // been padded a few bytes, we may end up writing out a few too
  // many bytes to the file (which may cause problems with some apps).
  //
  // So, instead let's calculate the size manually (if we can)
  //
  // First, find size of header plus size of color table.  Since the
  // first DWORD in both BITMAPINFOHEADER and BITMAPCOREHEADER conains
  // the size of the structure, let's use this.

  dwDIBSize = *(LPDWORD)lpBI + ::PaletteSize((LPSTR)lpBI);  // Partial Calculation

  // Now calculate the size of the image

  if ((lpBI->biCompression == BI_RLE8) || (lpBI->biCompression == BI_RLE4))
  {
      // It's an RLE bitmap, we can't calculate size, so trust the
      // biSizeImage field

      dwDIBSize += lpBI->biSizeImage;
  }
  else
  {
      DWORD dwBmBitsSize;  // Size of Bitmap Bits only

      // It's not RLE, so size is Width (DWORD aligned) * Height

      dwBmBitsSize = WIDTHBYTES((lpBI->biWidth)*((DWORD)lpBI->biBitCount)) * lpBI->biHeight;

      dwDIBSize += dwBmBitsSize;

      // Now, since we have calculated the correct size, why don't we
      // fill in the biSizeImage field (this will fix any .BMP files which
      // have this field incorrect).

      lpBI->biSizeImage = dwBmBitsSize;
  }


  // Calculate the file size by adding the DIB size to sizeof(BITMAPFILEHEADER)

  bmfHdr.bfSize = dwDIBSize + sizeof(BITMAPFILEHEADER);
  bmfHdr.bfReserved1 = 0;
  bmfHdr.bfReserved2 = 0;

  /*
   * Now, calculate the offset the actual bitmap bits will be in
   * the file -- It's the Bitmap file header plus the DIB header,
   * plus the size of the color table.
   */
  bmfHdr.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) + lpBI->biSize
                                               + PaletteSize((LPSTR)lpBI);

  TRY
  {
      // Write the file header
      file.Write((LPSTR)&bmfHdr, sizeof(BITMAPFILEHEADER));
      //
      // Write the DIB header and the bits
      //
      file.WriteHuge(lpBI, dwDIBSize);
  }
  CATCH (CFileException, e)
  {
      ::GlobalUnlock((HGLOBAL) hDib);
      THROW_LAST();
  }
  END_CATCH

  ::GlobalUnlock((HGLOBAL) hDib);
```

```
    return TRUE;
}

/*************************************************************************

  Function:  ReadDIBFile (CFile&)

   Purpose:  Reads in the specified DIB file into a global chunk of
             memory.

   Returns:  A handle to a dib (hDIB) if successful.
             NULL if an error occurs.

  Comments:  BITMAPFILEHEADER is stripped off of the DIB.  Everything
             from the end of the BITMAPFILEHEADER structure on is
             returned in the global memory handle.

*************************************************************************/


HDIB WINAPI ReadDIBFile(CFile& file)
{
    BITMAPFILEHEADER bmfHeader;
    DWORD dwBitsSize;
    HDIB hDIB;
    LPSTR pDIB;

    /*
     * get length of DIB in bytes for use when reading
     */

    dwBitsSize = file.GetLength();

    /*
     * Go read the DIB file header and check if it's valid.
     */
    if ((file.Read((LPSTR)&bmfHeader, sizeof(bmfHeader)) !=
        sizeof(bmfHeader)) || (bmfHeader.bfType != DIB_HEADER_MARKER))
    {
        return NULL;
    }
    /*
     * Allocate memory for DIB
     */
    hDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, dwBitsSize);
    if (hDIB == 0)
    {
        return NULL;
    }
    pDIB = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);

    /*
     * Go read the bits.
     */
    if (file.ReadHuge(pDIB, dwBitsSize - sizeof(BITMAPFILEHEADER)) !=
        dwBitsSize - sizeof(BITMAPFILEHEADER) )
    {
        ::GlobalUnlock((HGLOBAL) hDIB);
        ::GlobalFree((HGLOBAL) hDIB);
        return NULL;
    }
    ::GlobalUnlock((HGLOBAL) hDIB);
    return hDIB;
}
```

PACKMSG.CPP

```
/*************************************************************************\
 * FILE: PackMsg.cpp                                                     *
 *                                                                       *
 * DESCRIPTION:                                                          *
 * The PackedMsg class is responsible for creating an efficient binary  *
 * coding representation of the ASCII message the user wishes to embed  *
 * in the image.  This representation is "efficient" in that it packs   *
 * the message into a format which requires fewer total bits than that  *
 * used by the equivalent ASCII representation.                         *
 *                                                                       *
 * Currently, the packing scheme translates each ASCII character of the *
 * user message to a value which can be represented with 6 bits.  Some  *
 * ASCII characters have no representation, of course, since only 64   *
 * alphanumeric and special characters can be represented by the 6 bit *
 * code.  See the enumeration in the Packmsg.h file for the exact       *
 * translations used.                                                    *
 *                                                                       *
 * This C++ file contains the implementation code for the class.        *
 *                                                                       *
 * CREATION DATE:  August 31, 1995                                       *
 *                                                                       *
 * Copyright (c) 1995 Digimarc Incorporated, all rights reserved.*
\*************************************************************************/
#include "stdafx.h"
#include "packmsg.h"
#include <string.h>
#include <ctype.h>

typedef char * Compact_Msg;


///////////////////////////////////////////////////////////////
// PackedMsg(const char *user_msg)
//
// This is the PackedMsg constructor which is given an ASCII
// message for use by the signer.  It creates an array of
// packed characters (a more compact representation than
// ASCII), computes the checksum for the compact string,
// and then creates a bit array containing the compact
// message (this is the form the signer core algorithms
// require).
///////////////////////////////////////////////////////////////
PackedMsg::PackedMsg(const char *user_msg)
{
    m_correctBits = 0;
    m_checksum = 0;
    m_recoveredChecksum = 0;
    m_computedReaderChecksum = 0;


    // Save the  length, and a copy of the original user (ascii) message.
    m_msgLength = strlen(user_msg);
    m_asciiMsg = new char[m_msgLength+1];
    strcpy(m_asciiMsg, user_msg);              // Note it is null terminated.
    m_recoveredAsciiMsg = new char[m_msgLength+1];

    // Allocate space for the packed message.  Note there's no NULL termination.
    m_compactMsg = new char[m_msgLength];

    // Call the  function which translates to compact form.
    PackMessage();

    // Compute the checksum of the compact message string
    m_checksum = ComputeChecksum(m_compactMsg, m_msgLength);

    // Allocate space for the MsgBitArray, which puts one bit of the
    // packed message in each char of an unsigned char array (this is
    // the format that the current core signer needs.
    // Also, we include space for checksum of same length as 1 char.
    // Also allocate space for the ReaderBitArray, which reader will use.
    m_msgBitArrayLength = (m_msgLength+1) * PACKED_BITS_PER_CHAR;
    m_msgBitArray = new unsigned char[m_msgBitArrayLength];
    m_readerBitArray = new unsigned char[m_msgBitArrayLength];

    unsigned char *p_bit_array = m_msgBitArray;
    unsigned char *p_reader_array = m_readerBitArray;
    int i, j;
    unsigned char mask;
    for (i = 0; i < m_msgLength; i++)
    {
        for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
        {
            mask = 1 << j;
            if (m_compactMsg[i] & mask)
                *p_bit_array = 1;
            else
                *p_bit_array = 0;

            p_bit_array++;
            *p_reader_array++ = 0;          // clear the readers array.
        }
    }

    // Continue be putting the checksum in the final PACKED_BITS_PER_CHAR
    // elements of the bit array.
    for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
    {
        mask = 1 << j;
        if (m_checksum & mask)
            *p_bit_array = 1;
        else
            *p_bit_array = 0;

        p_bit_array++;
        *p_reader_array++ = 0;              // clear the readers array.
    }
}

// The PackedMsg constructor which is the length of a message to be read.
```

```
PackedMsg::PackedMsg(int msg_length)
{
  int i;

  m_correctBits = 0;

  // Save the  length, and allocate space for the ASCII message.
  m_msgLength = msg_length;
  m_asciiMsg = new char[m_msgLength+1];

  //  Null out the ascii storage
  for (i = 0; i < m_msgLength+1; i++)
      m_asciiMsg[i] = '\0';

  // Allocate space for the packed message.  Note there's no NULL termination.
  m_compactMsg = new char[m_msgLength];


  // Allocate space for the MsgBitArray, which will hold one bit of the
  // packed message in each char of an unsigned char array (this is
  // the format that the current core signer needs.
  // Also, we include space for checksum of same length as 1 char.
  // Also allocate space for the ReaderBitArray, which reader will use.
  m_msgBitArrayLength = (m_msgLength+1) * PACKED_BITS_PER_CHAR;
  m_msgBitArray = new unsigned char[m_msgBitArrayLength];
  m_readerBitArray = new unsigned char[m_msgBitArrayLength];

}

// The Destructor
PackedMsg::~PackedMsg()
{
  delete [] m_asciiMsg;
  delete [] m_compactMsg;
  delete [] m_msgBitArray;
  delete [] m_readerBitArray;
  delete [] m_recoveredAsciiMsg;
}


/////////////////////////////////////////////////////////////////
// PackMessage()
//
// Converts the ASCII message into an array of "packed" char-
// acters (currently 6 bits per packed character) which require
// a minimum of bandwidth in the Digimarc signed image.
/////////////////////////////////////////////////////////////////
void PackedMsg::PackMessage(void)
{
  int         i;
  char        ascii_ch;

  for (i = 0; i < m_msgLength; i++)
  {
      ascii_ch = toupper(m_asciiMsg[i]);

      if (ascii_ch >= '0' && ascii_ch <= '9')
          m_compactMsg[i] = zero + (ascii_ch - '0');

      else if (ascii_ch >= 'A' && ascii_ch <= 'Z')
      {
          m_compactMsg[i] = A + (ascii_ch - 'A');
      }

      // Check for special characters and encode them.
      else switch (ascii_ch)
      {
          case ' ':   m_compactMsg[i] = space;
                      break;
          case '.':   m_compactMsg[i] = period;
                      break;
          case ',':   m_compactMsg[i] = comma;
                      break;
          case ':':   m_compactMsg[i] = colon;
                      break;
          case '/':   m_compactMsg[i] = slash;
                      break;
          case '\\':  m_compactMsg[i] = backslash;
                      break;

          default:    m_compactMsg[i] = undefined;
                      // Warn user that an undefined character was found.
                      CString warn_msg;
                      warn_msg = "Sorry, but \"";
                      warn_msg += CString(ascii_ch);
                      warn_msg += "\" is not part of the Digimarc character set.";
                      warn_msg += "\nIt will be replaced by a '?'.";
                      MessageBox(NULL, warn_msg,
                              "Warning", MB_ICONINFORMATION | MB_OK);
                      break;
                }
        }
}
/////////////////////////////////////////////////////////////////
//
// BitsToString()
//
// Function which reads the recovered bit array, containing one bit of
// the packed binary message in each char element, and packs these bits
// into the m_compactMsg array (which then contains one packed msg
// character per element).  It then converts the compactMsg to
// ASCII and puts the resulting characters in the m_recoveredAsciiMsg
// array.  Also, the last PACKED_BITS_PER_CHAR bits contain the checksum.
// This is recovered and stored in the m_recoveredChecksum variable.
/////////////////////////////////////////////////////////////////
void PackedMsg::BitsToString(void)
{
    unsigned char *p_read_bits, *p_signed_bits;
    int i, j;
    unsigned char bit;

    // First, build the m_compactMsg array from the m_readerBitArray.

    //bit_array_ptr = m_readerBitArray;
    p_read_bits = m_readerBitArray;
    p_signed_bits = m_msgBitArray;
    m_correctBits = 0;
    for (i = 0; i < m_msgLength; i++)
    {
        m_compactMsg[i] = 0;    // Start with nothing.

        for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
        {
            if (*p_read_bits == 1)
            {
                bit = 1;
                m_compactMsg[i] |= (bit << j);
            }

            // Compute bit success rate metric:
            if (*p_read_bits == *p_signed_bits)
                m_correctBits++;

            p_read_bits++;
            p_signed_bits++;
        }
    }


    // Now recover the checksum from the end of the bit array.
    m_recoveredChecksum = 0;
    for (j = PACKED_BITS_PER_CHAR - 1; j >= 0; j--)
    {
        if (*p_read_bits == 1)
        {
            m_recoveredChecksum |= (1 << j);
        }

        // Compute bit success rate metric:
        if (*p_read_bits == *p_signed_bits)
            m_correctBits++;

        p_read_bits++;
        p_signed_bits++;
    }


    // Next, convert the compact form to an ASCII string.
    for (i = 0; i < m_msgLength; i++)
    {
        if (m_compactMsg[i] >= zero && m_compactMsg[i] <= nine)
            m_recoveredAsciiMsg[i] = '0' + m_compactMsg[i] - zero;

        else if (m_compactMsg[i] >= A && m_compactMsg[i] <= Z)
            m_recoveredAsciiMsg[i] = 'A' + m_compactMsg[i] - A;

        else switch (m_compactMsg[i])
        {
            case space:
                m_recoveredAsciiMsg[i] = ' ';
                break;
            case period:
                m_recoveredAsciiMsg[i] = '.';
                break;
            case comma:
                m_recoveredAsciiMsg[i] = ',';
                break;
            case colon:
                m_recoveredAsciiMsg[i] = ':';
```

```
            break;
        case slash:
            m_recoveredAsciiMsg[i] = '/';
            break;
        case backslash:
            m_recoveredAsciiMsg[i] = '\\';
            break;

        default:
            m_recoveredAsciiMsg[i] = '?';   // When we don't recognize the character.
            break;
        }
    }
    // Add a Null terminator
    m_recoveredAsciiMsg[m_msgLength] = '\0';

    // Compute the checksum of the read message
    m_computedReaderChecksum = ComputeChecksum(m_compactMsg, m_msgLength);
}

///////////////////////////////////////////////////////////////////////
//
// ComputeChecksum()
//
// This function is passed a pointer to the compact message
// string containing a message.  It computes and returns the checksum.
// The checksum algorithm used is a simple "spiral add", and the
// size of the checksum is PACKED_BITS_PER_CHAR (although it is
// stored as an unsigned char).
//
// NOTE:
// There is an implicit assumption that PACKED_BITS_PER_CHAR < 8
// If this changes, mods will be needed in this code.
///////////////////////////////////////////////////////////////////////
unsigned char PackedMsg::ComputeChecksum(char *pMsg, int length)
{
  int                  i;
  unsigned char        csum = 0;
  const unsigned char carry_bit_mask = (1 << PACKED_BITS_PER_CHAR);
  const unsigned char remove_carry_bit_mask = ~carry_bit_mask;

  for (i = 0; i < length; i++)
  {
      // Rotate the checksum: shift left and OR in the carry bit.
      csum = csum << 1;
      if (csum & carry_bit_mask)
      {
          csum |= 1;
          csum &= remove_carry_bit_mask;
      }

      // Add the next character
      csum += (unsigned char) *pMsg;

      // We want an unsigned add of length PACKED_BITS_PER_CHAR,
      // so remove the carry bit if its there.
        csum &= remove_carry_bit_mask;

        pMsg++;
    }

    return csum;
}
```

PACKMSG.H

```
/*******************************************************************\
* FILE: PackMsg.h                                                   *
*                                                                   *
* DESCRIPTION:                                                      *
* The PackedMsg class is responsible for creating an efficient binary *
* coding representation of the ASCII message the user wishes to embed *
* in the image.  This representation is "efficient" in that it packs *
* the message into a format which requires fewer total bits than that *
* used by the equivalent ASCII representation.                      *
*                                                                   *
* This header file should be included by any module which creates or *
* makes use of PackedMsg objects.                                   *
*                                                                   *
* CREATION DATE:  August 16, 1995                                   *
*                                                                   *
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.    *
\*******************************************************************/
#ifndef PACKMSG_H
#define PACKMSG_H
```

```
//#include "digimarc.h"
//#include "Params.h"

#define PACKED_BITS_PER_CHAR    6   // We will use 6 bits per user character

// We're going to use a 6 bit representation of up to 64 alphanumeric
// plus special characters.  The following enumeration indicates how
// each will be represented.  There first item takes value 0, 2nd item
// takes 1, ...
enum PackedChar
    {zero, one, two, three, four, five, six, seven, eight, nine,
     A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,
     space, period, comma, colon, slash, backslash,
     undefined};


typedef char * Compact_Msg;

class PackedMsg
{

// Public member functions
public:
    // Constructor: takes user's input message and creates the packed version.
    PackedMsg(const char *user_msg);

    // A Constructor for use by the reader.
    PackedMsg(int msg_length);

    // An accessor allows callers read-only access to the packed msg.
    const Compact_Msg getCompactMsg(void) const;
    int getCompactMsgSize(void) const;
    unsigned char *getMsgBitArray(void) const {return m_msgBitArray;}
    int getMsgBitArrayLength(void) const {return m_msgBitArrayLength;}
    char *getAsciiMsg(void) const {return m_asciiMsg;}
    unsigned char *getReaderBitArray(void) const {return m_readerBitArray;}
    char *getRecoveredAsciiMsg(void) const {return m_recoveredAsciiMsg;}

    int GetNumCorrectBits(void) const {return m_correctBits;}
    float GetPercentCorrect(void) const
        {return (float) m_correctBits * (float)100.0 / (float) m_msgBitArrayLength;}

    // Checksum accessors.
    unsigned char GetSignerChecksum(void) {return m_checksum;}
    unsigned char GetReaderChecksum(void) {return m_recoveredChecksum;}
    unsigned char GetComputedReaderChecksum(void) {return m_computedReaderChecksum;}

    int GetMsgLength(void) const {return m_msgLength;}

    // Function to unpack a message, for use by the recognizer...
    void BitsToString(void);

    // Destuctor
    ~PackedMsg(void);


// Private member functions
private:
    void PackMessage(void);
    unsigned char ComputeChecksum(char *pMsg, int length);

// Private data
private:
    char            *m_asciiMsg;        // The original ASCII message  ASCII(null terminated).
    int             m_msgLength;        // No. of chars (not included null terminator.
    Compact_Msg     m_compactMsg;       // The message in the packed format.

    unsigned char   *m_msgBitArray;     // Core signer algorithm wants one bit per char.
                                        // Includes checksum.
    int             m_msgBitArrayLength;
    unsigned char   *m_readerBitArray;  // Array of bits recovered by reader,
                                        // includes checksum.
    char            *m_recoveredAsciiMsg;//The recovered message

    unsigned char   m_checksum;
    unsigned char   m_recoveredChecksum;
    unsigned char   m_computedReaderChecksum;

    int             m_correctBits;
};

#endif  // PACKMSG_H
```

PARAMS.CPP

```
/*******************************************************************\
```

```
* FILE: Params.cpp                                                    *
*                                                                     *
* DESCRIPTION:                                                        *
* Implementation of the Parameters classes: SignerParams and          *
* ReaderParams.                                                       *
*                                                                     *
*                                                                     *
* CREATION DATE:  September 8, 1995                                   *
*                                                                     *
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.      *
\*********************************************************************/

#include "params.h"
#include "stdafx.h"
#include <string.h>
#include <strstrea.h>


//////////////////////////////////////////////////
// CONSTRUCTOR FOR SIGNER PARAMS OBJECT WHICH
// TAKES THE COMMAND LINE STRING AS AN ARGUMENT.
//////////////////////////////////////////////////


SignerParams::SignerParams(LPSTR cmd_line)              // Constructor based on command line
{
  char *dash_ptr, *cmd_type, *cmd, *commands;
  const char *dbg_msg_ptr;

  parameters.input_filename = NULL;
  parameters.message = "Default Message";
  parameters.output_filename = NULL;
  parameters.registry_name = NULL;

  parameters.user_key = 1;
  parameters.gain = (float) 100.0;
  parameters.gamma = (float) 0.07;

  parameters.bump_size = 1;

  parameters.lut_scale = (float) 100.0;

  parameters.super_reader_flag = FALSE;

  dbg_msg_ptr = (const char *) GetMessage();

  TRACE("Debug in SignerParams constructor.  Message is: %s\n", dbg_msg_ptr);

  // Make a copy of the command line that we can mutilate
  commands = new char[strlen(cmd_line) + 1];
  strcpy(commands, cmd_line);

  dash_ptr = NULL;

  // If the command line doesn't start w/ a '-', then the command line is
  // a single argument: the filename.  This case comes up when the program
  // is invoked by dragging a filename onto the executable in Win95 explorer.
  if (strlen(cmd_line) > 0 && cmd_line[0] != '-')
  {
      parameters.input_filename = new char[strlen(cmd_line) +1];
      strcpy(parameters.input_filename, cmd_line);
  }
  // Otherwise, we check for the multiple argument format of the command line,
  // in which arguments pairs are used, e.g., "-f <filename>".
  else
  {
      do
      {
          // Find the last '-' character
          dash_ptr = strrchr(cmd_line, '-');

          if (dash_ptr != NULL)
          {
              cmd_type = dash_ptr + 1;
              cmd = cmd_type + 1;

              // Create an in-core input stream
              istrstream inStream(cmd, strlen(cmd));

              switch (*cmd_type)
              {
              case 'g':
              case 'G':
                  inStream >> parameters.gain;
                  break;
              case 'f':
              case 'F':
                  parameters.input_filename = new char[strlen(cmd) +1];
                  inStream >> parameters.input_filename;
                  break;
              case 'm':
              case 'M':
              //  parameters.message = new char[strlen(cmd) +1];
              //  inStream.getline(parameters.message,
              //                   strlen(cmd)+1,
              //                   '\0');
                  parameters.message = cmd;
              case 'z':
              case 'Z':
                  inStream >> parameters.gamma;
              default:
                  break;
              }
              // Lop off the last argument by replacing the dash with a NULL;
              *dash_ptr = '\0';
          }
      } while (dash_ptr != NULL);
  }

  //if (parameters.message == NULL)
  //{
  //  parameters.message = new char[strlen("Default message") + 1];
  //  strcpy(parameters.message, "Default message");
  //}


  // Clean up.
  delete [] commands;
}


SignerParams::~SignerParams(void)
{
    if (parameters.input_filename != NULL)
        delete [] parameters.input_filename;

    //if (parameters.message != NULL)
    //  delete [] parameters.message;

    if (parameters.output_filename != NULL)
        delete [] parameters.output_filename;

    if (parameters.registry_name != NULL)
        delete [] parameters.registry_name;
}

/////////////////////////////////////////////////////////////
// SignerParams::UpdateSignTime()
//
// Update the timestamp member variable within this object.
/////////////////////////////////////////////////////////////
void SignerParams::UpdateSignTime(void)
{
    // Set the timestamp indicating when we signed this puppy.


    CTime t = CTime::GetCurrentTime();

    parameters.sign_time = t;
}
```

PARAMS.H

```
/*********************************************************************\
* FILE: Params.h                                                      *
*                                                                     *
* DESCRIPTION:                                                        *
* The Params classes are responsible for gathering and managing all   *
* user input parmaters.  There are two classes defined here: 1) the*
* SignerParams class for the signer, and the ReaderParams class for the *
* reader.                                                             *
*                                                                     *
* The SignerParams class also keeps track of internal parameters which *
* control or "tune" the operation of the signer, but which are not*
* accessible by the user.                                             *
*                                                                     *
* At present, this is a non-GUI version.  All*
* user inputs enter from the command line.  In the future, a GUI version*
* will be added which will present a dialog box to the user and gather *
* input parameters from a graphical interface.  The command line version*
* will probably always exist for testing purposes and possibly batch*
* processing.  Different constructors will be used to differentiate *
* between the GUI and cmd line versions.                              *
*                                                                     *
* This header file should be included by any module which creates or*
* makes use of SignerParams and/or ReaderParams objects.*
*                                                                     *
```

```
* CREATION DATE:  August 15, 1995                                    *
*                                                                    *
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.    *
\*******************************************************************/
#ifndef PARAMS_H
#define PARAMS_H

// #include "digimarc.h"
#include <time.h>
#include "stdafx.h"


/////////////////////////////////////////////////
// SIGNER PARAMETERS STRUCTURES AND CLASSES
/////////////////////////////////////////////////

typedef unsigned User_key_t;

// Define a structure which will contain the various Signer  parameters.
// The Signer Params class will contain a private copy of this structure.
typedef struct
{
  // User inputs...
  char          *input_filename;
  CString       message;
  User_key_t    user_key;
  char          *output_filename;
  char          *registry_name;

  // "Super user" inputs, useful for testing and tuning, go here.
  float         gain;
  float         gamma;
  int           bump_size;
  float         lut_scale;
  BOOL          super_reader_flag;

  // Non user inputs will go here...

  // Some parameters which indicate what happened during use...
  CTime         sign_time;
} signer_param_struct;

// TBD: create a Params virtual base class for use by signer and reader params.

class SignerParams
{
// Public member functions and data structures
public:
  SignerParams(LPSTR cmd_line);                // Constructor based on command line
//    SignerParams(signer_param_struct *params);  // Constructor used during reading, based
                                               // on reading the registry.
  ~SignerParams(void);

  void UpdateSignTime(void);

  // Create an accessor which returns a ptr to a const copy of the parameters stucture.
  // An alternative is to write accessors for each individual parameter.
  const signer_param_struct * getParams(void) const;

  // Accessors for specific parameters...
  float             GetGain(void) {return parameters.gain;}
  void              SetGain(float newgain) {parameters.gain = newgain;}
  float             GetGamma(void) {return parameters.gamma; }
  void              SetGamma(float newgamma) {parameters.gamma = newgamma;}
  char              *GetInputFilename(void) {return parameters.input_filename;}
  const CString&    GetMessage(void) {return parameters.message;}
  void              SetMessage(CString& newstring) {parameters.message = newstring;}
  UINT              GetKey(void) {return (UINT) parameters.user_key;}
  void              SetKey(UINT newkey) {parameters.user_key = newkey;}
  const CTime&      GetTimestamp(void) {return parameters.sign_time;}
  BOOL              GetSuperReaderFlag(void) {return parameters.super_reader_flag;}
  void              SetSuperReaderFlag(BOOL newflag)
                        {parameters.super_reader_flag = newflag;}
  int               GetBumpSize(void) {return parameters.bump_size;}
  void              SetBumpSize(int size) {parameters.bump_size = size;}
  float             GetLutScale(void) {return parameters.lut_scale;}
  void              SetLutScale(float new_scale) {parameters.lut_scale = new_scale;}


// Private member functions and data structures
private:
  signer_param_struct parameters;         // stucture containing the user parameters.

  // Function which warns user if parameters are not all present or look incorrect.
  // It will also throw an exception if things are not right.
  checkParams(void);

};
```

```
/////////////////////////////////////////////////
// READER PARAMETERS STRUCTURES AND CLASSES
/////////////////////////////////////////////////

// Define a structure which will contain the various Reader parameters.
// The Reader Params class will contain a private copy of this structure.
typedef struct
{
    // User inputs...
    char        *input_filename;

    // User provides some combination of following to uniquely locate
    // the registry entry for the signing event...
    User_key_t  user_key;
    time_t      date_of_signing;

    char        *registry_name;         // optional

    // "Super user" inputs, useful for testing and tuning, go here.


    // Non user inputs will go here...

} reader_param_struct;


class ReaderParams
{
// Public member functions and data structures
public:
    ReaderParams(int argc, char *argv[]);   // Constuctor for non-gui (cmd line) version

    // Create an accessor which returns a ptr to a const copy of the parameters stucture.
    // An alternative is to write accessors for each individual paramoter.
    const reader_param_struct * getParams(void) const;



// Private member functions and data structures
private:
    reader_param_struct parameters;         // stucture containing the user parameters.

    // Function which warns user if parameters are not all present or look incorrect.
    // It will also throw an exception if things are not right.
    checkParams(void);

};

#endif  // PARAMS_H
```

PARMSDLG.CPP

```
// parmsdlg.cpp : implementation file
//

#include "stdafx.h"
#include "signer.h"
#include "parmsdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// ParmsDlg dialog


ParmsDlg::ParmsDlg(CWnd* pParent /*=NULL*/)
    : CDialog(ParmsDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(ParmsDlg)
    m_message = "";
    m_gain_from_edit_box = (float) 0.0;
    m_key = 0;
    m_bump_size = 0;
    m_detail_lut_scale = 0.0f;
    //}}AFX_DATA_INIT
}

void ParmsDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(ParmsDlg)
    DDX_Text(pDX, IDC_MESSAGE, m_message);
    DDV_MaxChars(pDX, m_message, 256);
```

```
    DDX_Text(pDX, IDC_EDIT_GAIN, m_gain_from_edit_box);
    DDV_MinMaxFloat(pDX, m_gain_from_edit_box, 1.e-003f, 1.e+006f);
    DDX_Text(pDX, IDC_EDIT_KEY, m_key);
    DDX_Text(pDX, IDC_BUMP_SIZE, m_bump_size);
    DDV_MinMaxInt(pDX, m_bump_size, 1, 256);
    DDX_Text(pDX, IDC_DETAIL_SCALE, m_detail_lut_scale);
    DDV_MinMaxFloat(pDX, m_detail_lut_scale, 1.e-003f, 1.e+006f);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(ParmsDlg, CDialog)
  //{{AFX_MSG_MAP(ParmsDlg)
  ON_COMMAND(ID_SETTINGS_SIGNER, OnSettingsSigner)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// ParmsDlg message handlers

void ParmsDlg::OnOK()
{

  CDialog::OnOK();
}

void ParmsDlg::OnSettingsSigner()
{
  // TODO: Add your command handler code here

}
```

PARMSDLG.H

```
// parmsdlg.h : header file
//
#include "stdafx.h"
/////////////////////////////////////////////////////////////////////////////
// ParmsDlg dialog

class ParmsDlg : public CDialog
{
// Construction
public:
  ParmsDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
  //{{AFX_DATA(ParmsDlg)
  enum { IDD = IDD_PARAMS_DIALOG };
  CString m_message;
  float   m_gain_from_edit_box;
  UINT    m_key;
  int     m_bump_size;
  float   m_detail_lut_scale;
  //}}AFX_DATA

// Implementation
protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

  // Generated message map functions
  //{{AFX_MSG(ParmsDlg)
  virtual void OnOK();
  afx_msg void OnSettingsSigner();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};
```

RAWIMAGE.H

```
/*********************************************************************\
* FILE: RawImage.h                                                    *
*                                                                     *
* DESCRIPTION:                                                        *
* RawImage objects are used to convert images from popular formats    *
* to the raw image format used internally by the Digimarc system.    *
* Typically, the RawImage constructor is given an input file as an    *
* argument, and the constuctor is responsible for reading the file    *
* and performing the necessary operations to convert it into the raw *
* format.                                                             *
*                                                                     *
* RawImage objects also are able to perform the inverse conversion,  *
* creating image files in various standard formats from the internal *
* raw representation.                                                 *
*                                                                     *
* The initial implementation will only except TIFF files as inputs,  *
* and will make use of the public domain software LibTiff in order*
* to read and write TIFF files.                                       *
*                                                                     *
* This header file should be included by any module which creates or*
* makes use of RawImage objects.                                      *
*                                                                     *
* CREATION DATE:    August 15, 1995                                   *
*                                                                     *
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.*
\*********************************************************************/
#ifndef RAWIMAGE_H
#define RAWIMAGE_H

#include "digimarc.h"
#include "Params.h"


// Since the exact internal representation may change, use a typedef.
// This will allow a single change to modify all references to the
// raw image data format.
// Also note that in the future we will need several raw image representation.
typedef long * Raw_Data;


class RawImage
{
// Public member functions and data structures
public:
    RawImage(SignerParams *params);

    // Member function which gives caller access to the raw image and its attibutes.
    const int getXdim(void);
    const int getYdim(void);

    // This accessor returns a const pointer to a read-only image.
    const Raw_Data getImage(void) const;

    // This accessor returns a const pointer to a writable image.
    Raw_Data * getWritableImage(void) const;


    //Member function used to convert the raw image to an output TIFF file.
    writeTiff(char *filename);



// Private data.  Users of rawImage objects get at these through accessors only.
private:
    int         xdim;       // X dimension of image
    int         ydim;       // Y dimension of image
    Raw_Data    image;      // Ptr to array of image data

};

#endif  // RAWIMAGE_H
```

READ.CPP

```
/////////////////////////////////////////////////////////////////////////////
// FILE: Read.dpp
//
// DESCRIPTION:
// Core recognition functions of the Digimarc technology
// Created August 1995
//
// This particular code uses "raster" based processing as opposed to 2D based
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
/////////////////////////////////////////////////////////////////////////////
#include "read.h"
#include "sign.h"
#include "fft.h"
#include "stdafx.h"
#include <math.h>

/* Constants */
const float epsilon = (float) 0.000001;

/////////////////////////////////////////////////////////////////////////////
// read_8bit_single_channel_or_color()
//
// Used to read (or "recognize") the embedded digimarc signature in
// either a gray-scale or color image.  Set number_channels to 1 for
// gray-scale, 3 for color.
/////////////////////////////////////////////////////////////////////////////
int read_8bit_single_channel_or_color(
    unsigned char *data,            /* input data to be recognized */
    long original_xdim,             /* it's x dimension */
```

```
    long original_ydim,            /* it's y dimension */
    long x_offset,                 /* x offset of segment */
    long y_offset,                 /* y offset of segment */
    long x_extent,                 /* x extent of segment */
    long y_extent,                 /* y extent of segment */
    int message_length,            /* length of message in BITS, also length of message string */
    unsigned char *key,            /* original 8 bit random key */
    long key_length,               /* key_length often equal to data_length but not always */
/**unused**/
    char *key_lut,                 /* look up table mapping key value */
    float *luminance_lut,          /* look up table mapping the signature level to luminance*/
    float *detail_lut,             /* look up table mapping the signature level to local detail*/

    unsigned char *thumbnail,      /* if available, use pointer, otherwise NULL*/
    unsigned char *original_data,  /* if available, use pointer, otherwise NULL*/

      const unsigned char *referenceBitArray,   // bit array ptr: either the known message or estimate.
    float *metric,                 // we will compute a return a crude metric indicating confidence.
    float *range,
    unsigned char *message,        /* output: either 0 or 1, i.e. inefficient but simple */
    int number_channels,            // generally for B&W==1 vs. color == 3
    int reading_mode,
    int bumps
){
    int status = 1;


    if(reading_mode == 0){

        read_8bit_single_channel_OLD_plus_color(
            data, original_xdim, original_ydim, x_offset, y_offset,
            x_extent, y_extent, message_length, key, key_length, key_lut,
            luminance_lut, detail_lut, thumbnail, original_data, referenceBitArray,
            metric, range, message, number_channels,bumps);
    }

    else if(reading_mode == 1){

        read_super(
            data, original_xdim, original_ydim, x_offset, y_offset,
            x_extent, y_extent, message_length, key, key_length, key_lut,
            luminance_lut, detail_lut, thumbnail, original_data, referenceBitArray,
            metric, range, message, number_channels,bumps);
    }


        return(status);
}


///////////////////////////////////////////////////////////////////////////
// read_8bit_single_channel_OLD_plus_color()
///////////////////////////////////////////////////////////////////////////
void read_8bit_single_channel_OLD_plus_color(
    unsigned char *data,           /* input data to be recognized */
    long original_xdim,            /* it's x dimension */
    long original_ydim,            /* it's y dimension */
    long x_offset,                 /* x offset of segment */
    long y_offset,                 /* y offset of segment */
    long x_extent,                 /* x extent of segment */
    long y_extent,                 /* y extent of segment */
    int message_length,            /* length of message in BITS, also length of message string */
    unsigned char *key,            /* original 8 bit random key */
    long key_length,               /* key_length often equal to data_length but not always */
/**unused**/
    char *key_lut,                 /* look up table mapping key value */
    float *luminance_lut,          /* look up table mapping the signature level to luminance*/
    float *detail_lut,             /* look up table mapping the signature level to luminance*/

    unsigned char *thumbnail,      /* if available, use pointer, otherwise NULL*/
    unsigned char *original_data,  /* if available, use pointer, otherwise NULL*/

      const unsigned char *referenceBitArray,   // bit array ptr: either the known message or estimate.
    float *metric,                 // we will compute a return a crude metric indicating confidence.
    float *range,
    unsigned char *message,        /* output: either 0 or 1, i.e. inefficient but simple */
    int number_channels,
    int bumps
){

    unsigned char *pkey,*pdata;
    long i, line, bit;
    int temp, status=1;
    float *key_value = new float[x_extent];
    float *data_float = new float[x_extent];
    float *orig_float = new float[x_extent];
    float *bit_total =  new float[message_length];
    //float *bit_mag = new float[message_length];
    float *pkey_value, *pdata_float;
    float filter_cf = (float)0.5; // kludge for now
    double maxdiff = 40.0; // kludge for now
    int key_xlength = 1+(original_xdim-1)/bumps;
    for(i=0; i<message_length; i++)
    {
        bit_total[i]= (float) 0.0;
        //bit_mag[i] = (float) 0.0;
    }

    pdata = data;
    for(line=y_offset; line<(y_offset+y_extent); line++)
    {
        /* FIRST:  If either the original image or a thumbnail of the original is available,
        then use either a simple or "advanced" dot product to remove it; "advanced" refers
        to the idea that you may wish to adjust the gamma or higher order stuff */
        float_it(pdata, data_float, x_extent,number_channels);
        //derivative_threshold(data_float, x_extent, number_channels,maxdiff,filter_cf);
        //remove_mean(data_float, x_extent);

        /* load key_values */
        int key_offset  = (line/bumps)*key_xlength;
        pkey = &key[key_offset + x_offset/bumps];
        pkey_value = key_value;
        if(bumps>1){
            for(i=x_offset;i<(x_offset+x_extent);i++){
                *(pkey_value++) = (float)( (int)key_lut[ (int)*pkey ] );
                if( !((i+1)%bumps) )pkey++;
            }
        }
        else {
            for(i=x_offset;i<(x_offset+x_extent);i++){
                *(pkey_value++) = (float)( (int)key_lut[ (int)*(pkey++) ] );
            }
        }
        pdata+=(number_channels*x_extent);

        /* now step through processed patch and perform simple or "advanced" correlation
detection,
            keeping the resultant detection values in the accumulators for each bit of the
message_length
            bits */
        pdata_float = data_float;
        pkey_value = key_value;
        float running_average = (float) 0.0;
        float ftemp;
        for (i = 0; i < MOV_AV_KERNEL; i++)
        {
            running_average += *(pdata_float++);
        }
        float mov_av = (float)MOV_AV_KERNEL;
        running_average /= mov_av;
        pdata_float = data_float;
        temp = MOV_AV_KERNEL/2;
        int temp1 = temp+1;
        if(bumps>1){
            for (i = x_offset; i < (x_offset + x_extent); i++)
            {
                if (i <= (x_offset + temp) || i >= (x_offset + x_extent - temp) );
                else
                {
                    ftemp = (*(pdata_float + temp) - *(pdata_float - temp1)) / mov_av;
                    running_average += ftemp;
                }
                bit = ( key_offset + i/bumps) % message_length;
                ftemp = *(pdata_float++) - running_average;
                //bit_mag[bit] += (*pkey_value * *pkey_value);
                bit_total[bit] += (ftemp * *(pkey_value++));
            }
        }
        else {
            /*
            for (i = x_offset; i < (x_offset + x_extent); i++)
            {
                if (i <= (x_offset + temp) || i >= (x_offset + x_extent - temp) );
                else
                {
                    ftemp = (*(pdata_float + temp) - *(pdata_float - temp1)) / (float)
MOV_AV_KERNEL;
                    running_average += ftemp;
                }
                bit = ( key_offset + i) % message_length;
                //bit_mag[bit] += (*pkey_value * *pkey_value);
                bit_total[bit] += ( ( *(pdata_float++) - running_average) * *(pkey_value++));
            }
            */
            // time optimized version of above earlier code
            int key_foo = key_offset + x_offset;
            for(i=x_offset;i<=(x_offset+temp);i++){
```

```
            bit = key_foo++ % message_length;
            bit_total[bit] += ( ( *(pdata_float++) - running_average) * *(pkey_value++));
        }
        int temp2 = x_offset + x_extent - temp;
        float *pdata_float2 = data_float;
        float *pdata_float1 = &pdata_float[temp];
        for(i=(x_offset+temp+1);i<temp2;i++){
            running_average += (( *(pdata_float1++) - *(pdata_float2++) )/mov_av);
            bit = key_foo++ % message_length;
            bit_total[bit] += ( ( *(pdata_float++) - running_average) * *(pkey_value++));
        }
        for(i=0;i<temp;i++){
            bit = key_foo++ % message_length;
            bit_total[bit] += ( ( *(pdata_float++) - running_average) * *(pkey_value++));
        }
      }
  }

   /* fill the message string based on bit_totals */

   for(i=0; i<message_length; i++)
   {
     if(bit_total[i]>0.0)
     {
        message[i]=1;
     }
       else
       {
        message[i]=0;
       }
   }

  /*
   for (i = 0; i < message_length; i++)
   {
      // Before normalizing by the magnitudes, be sure we aren't
      // dividing by zero (this happens for an image w/ zero energy.
      if (bit_mag[i] == (float)0.0)
          bit_mag[i] = epsilon;

     bit_total[i] /= (float) sqrt( (double) bit_mag[i] );
   }
  */

    // Compute the "crude metric", an estimate of rms spread of the
  // bit level detector's results.  The referenceBitArray is either
  // the known message (if it was available to caller) or the
  // knewly computed estimate of the message.

  *metric = get_crude_metric(referenceBitArray, bit_total, range, message_length);


    delete [] data_float;
    delete [] orig_float;
    delete [] bit_total;
    delete [] key_value;
    //delete [] bit_mag;

      return;
}

////////////////////////////////////////////////////////////////////////////
// float_it()
//
////////////////////////////////////////////////////////////////////////////
void float_it(unsigned char *data, float *data_float,
            long x_extent, int number_channels)
{
  unsigned char *pdata;
  long    i;
  float *pfdata;

  pdata = data;
  pfdata=data_float;
  if(number_channels == 1){
      for (i = 0; i < x_extent; i++)
          *(pfdata++) = (float) *(pdata++);
  }
  else if (number_channels == 3) {
      for (i = 0; i < x_extent; i++){
          *pfdata = (float) *(pdata++);
          *pfdata += (float) *(pdata++);
          *(pfdata++) += (float) *(pdata++);
      }
  }
}

////////////////////////////////////////////////////////////////////////////

// remove_mean()
//
////////////////////////////////////////////////////////////////////////////
void remove_mean(float *array, long length)
{
    long    i;
    float   total = (float) 0.0;

    for (i = 0; i < length; i++)
        total += array[i];

    total /= (float) length;

    for (i = 0; i < length; i++)
        array[i] -= total;
}


////////////////////////////////////////////////////////////////////////////
// get_crude_metric()
//
//
////////////////////////////////////////////////////////////////////////////
float get_crude_metric(
    const unsigned char *actual_message,    // the the original message, if you have it,
                                            // otherwise use found message
    float *bit_total,
    float *range,
    int message_length)
{
    int i;
    float avg = (float) 0.0, rms = (float) 0.0, ftemp;

    *range = (float) 0.0;

    // add up all the 1's to find an average, as well as 0's
    for(i=0; i<message_length; i++)
    {
        if (actual_message[i] > 0)
            avg += bit_total[i];
        else
            avg -= bit_total[i];
    }
    avg /= message_length;

    // For a zero energy image, avg will equal zero.  We replace it
    // with epsilon.
    if (avg == 0.0)
        avg = epsilon;

    for (i = 0; i < message_length; i++)
        bit_total[i] /= avg;

    // now calculate the deviation about the nominal averages
    for(i=0; i<message_length; i++)
    {
        if (actual_message[i] > 0)
            ftemp = bit_total[i] - (float) 1.0;
        else
            ftemp = bit_total[i] + (float) 1.0;

        if ( fabs( (double)ftemp) > (double) *range)
            *range = (float) fabs( (double) ftemp);

        rms += (ftemp * ftemp);
    }
    ftemp = rms/ ((float)message_length - (float) 1.0);
    rms = (float) sqrt(ftemp);

    return( rms);   /* returns crude spread metric  */
}


int derivative_threshold(float *data, long length, int number_channels,double maxdiff,float
filter_cf)
{
    long i;
    int status = 1;

    float *pdata,llast,last;
    double diff;

    float replacement = (float)0.0;

    if(number_channels == 3)maxdiff *= 3.0;

    last = llast = data[0];
    pdata = &data[1];
```

```
    for(i=1;i<length;i++){
        diff = (double)*pdata - last;
        last = *pdata;
        if( fabs(diff) > maxdiff ){
            if (diff>0.0) diff = replacement;
            else diff = -replacement;
        }
        *pdata = llast + (float)diff;
        llast = *(pdata++);
    }


    return(status);
}


void read_super(
  unsigned char *data,            /* input data to be recognized */
  long original_xdim,             /* it's x dimension */
  long original_ydim,             /* it's y dimension */
  long x_offset,                  /* x offset of segment */
  long y_offset,                  /* y offset of segment */
  long x_extent,                  /* x extent of segment */
  long y_extent,                  /* y extent of segment */
  int message_length,             /* length of message in BITS, also length of message string */
  unsigned char *key,             /* original 8 bit random key */
  long key_length,                /* key_length often equal to data_length but not always */
/**unused**/
  char *key_lut,                  /* look up table mapping key value */
  float *luminance_lut,           /* look up table mapping the signature level to luminance*/
  float *detail_lut,              /* look up table mapping the signature level to luminance*/

  unsigned char *thumbnail,       /* if available, use pointer, otherwise NULL*/
  unsigned char *original_data,   /* if available, use pointer, otherwise NULL*/

   const unsigned char *referenceBitArray,    // bit array ptr: either the known message or estimate.
  float *metric,                  // we will compute a return a crude metric indicating confidence.
  float *range,
  unsigned char *message,         /* output: either 0 or 1, i.e. inefficient but simple */
  int number_channels,
  int bumps
){

  unsigned char *pkey,*pdata;
  long i, line, bit;
  int status=1,bits,fftdim,j,highest;
  float *bit_total =  new float[message_length];
  float *bit_mag = new float[message_length];
  float *key_value = new float[x_extent],*pkey_value;

  int key_xlength = 1+(original_xdim-1)/bumps;

  for(i=0; i<message_length; i++)
  {
        bit_total[i]= (float) 0.0;
        bit_mag[i] = (float) 0.0;
    }

  // find power of 2 higher than highest dimension
  if(x_extent > y_extent)highest = x_extent;
  else highest = y_extent;
  bits = 1 + (int)( log( (double)highest - 0.5 ) / log(2.0) );
  fftdim = (int)pow(2.0,(double)bits + 0.00000001);

  // create array
  float *image =  new float[fftdim*(fftdim+2)];
  float *wr = new float[fftdim];
  float *wi = new float[fftdim];
  float *pimage;
  pimage = image;
  for(i=0;i<(fftdim*(fftdim+2));i++)*(pimage++) = (float)0.0;

  // convert either a B&W image or a color image to a single floating point luminance image
  float total;
  if(number_channels == 1){
      pdata = data;
      for(i=0;i<y_extent;i++){
          pimage = &image[i*fftdim];
          for(j=0;j<x_extent;j++){
              *pimage = (float)*(pdata++);
              total += *(pimage++);
          }
      }
  }
  else if(number_channels == 3){
      pdata = data;
      for(i=0;i<y_extent;i++){
          pimage = &image[i*fftdim];
          for(j=0;j<x_extent;j++){
                  *pimage = (float)*(pdata++);
                  *pimage += (float)*(pdata++);
                  *pimage += (float)*(pdata++);
                  total += *(pimage++);
              }
          }

          // weird derivative threshold
          int choo=0;
          if(choo){
          // remove dc
          total /= ((float)y_extent * (float)x_extent);
          for(i=0;i<y_extent;i++){
              pimage = &image[i*fftdim];
              for(j=0;j<x_extent;j++){
                      *(pimage++)-=total;
              }
          }


          float *pdetail_vector;
          float *detail_vector = new float[x_extent];
          int start = 5;
          int stop = 500;
          float scale = (float)0.5;
          for(i=0;i<y_extent;i++){

get_read_detail_vector(detail_vector,data,x_extent,i,y_extent,number_channels,start,stop,scale
,image,fftdim);
              pdetail_vector = detail_vector;
              pimage = &image[i*fftdim];
              for(j=0;j<x_extent;j++)*(pimage++) += *(pdetail_vector++);
          }
          delete [] detail_vector;
          }


          //float filter_cf = (float)0.5; // kludge for now
          //double maxdiff = 40.0; // kludge for now
          //for(line=0; line<y_extent; line++)
          //{
          //  derivative_threshold(&image[line*fftdim], x_extent,1,maxdiff,filter_cf);
          //}

          // easy does the window ??
          // for now, multiply the last four values near the edges by a linear ramp to zero, simply
to avoid total edge weirdnesses
          int window_it=0;
          if(window_it){
          if(x_extent > 10 && y_extent > 10){
              float mult[4],*pmult;
              mult[0]=(float)0.2;mult[1]=(float)0.4;mult[2]=(float)0.6;mult[3]=(float)0.8;
              pmult = mult;
              for(i=1;i<5;i++){
                  pimage = &image[(i-1)*fftdim];
                  for(j=0;j<x_extent;j++)*(pimage++) *= *pmult;
                  pmult++;
              }
              pmult = mult;
              for(i=1;i<5;i++){
                  pimage = &image[(y_extent - i)*fftdim];
                  for(j=0;j<x_extent;j++)*(pimage++) *= *pmult;
                  pmult++;
              }
              for(i=0;i<y_extent;i++){
                  pimage = &image[i*fftdim];
                  pmult = mult;
                  for(j=1;j<5;j++)*(pimage++) *= *(pmult++);
                  pimage = &image[(i+1)*fftdim-(fftdim-x_extent+1)];
                  pmult = mult;
                  for(j=1;j<5;j++)*(pimage--) *= *(pmult++);
              }
          }
          }

          // fft arrays
          realfft2d_in_place(image,bits,0,wr,wi);

          // filter them
          // phase difference only to start
          /* calculate phase differences and reload them into real1 and imaginary1 */
          float mag1,*preal1,*pimaginary1;
          // double power = 0.8;
          preal1=image;pimaginary1=&image[fftdim];
          for(i=0;i<(1+fftdim/2);i++){
              for(j=0;j<fftdim;j++){
                  mag1 = (float)fabs( (double)*preal1 ) +  (float)fabs( (double)*pimaginary1 );
                  if(mag1 == (float)0.0){
                      *(preal1++) =(float)0.0;
```

```
            *(pimaginary1++) = (float)0.0;
        }
        else {
            //mag1 = (float)pow((double)mag1,power);
            *(preal1++) /= mag1;
            *(pimaginary1++) /= mag1;
        }
    }
    preal1+=fftdim;
    pimaginary1+=fftdim;
}

// remove low and/or high frequencies
// the DC should reside at row one, fftdim/2
int moo = 0;
if(moo){
int low = 1;
int xcount=low*2-1;
pimage = &image[(fftdim/2) - low +1];
for(i=0;i<2*low;i++){
    for(j=0;j<xcount;j++)*(pimage++) = (float)0.0;
    pimage += (fftdim - xcount);
}
}

// inverse fft
realfft2d_in_place(image,bits,1,wr,wi);

for(line=y_offset; line<(y_offset+y_extent); line++)
{

    /* load key_values */
    pkey = &key[(line/bumps) * key_xlength + x_offset/bumps];
   for(i=x_offset;i<(x_offset+x_extent);i++){
        key_value[i-x_offset] = (float)( (int)key_lut[ (int)*pkey ] );
        if( (i+1)%bumps )pkey++;
   }

    /* now step through processed patch and perform simple or "advanced" correlation detection,
       keeping the resultant detection values in the accumulators for each bit of the
message_length
       bits */
    pimage = &image[(line-y_offset)*fftdim];
    pkey_value = key_value;
    for(i=x_offset;i<(x_offset+x_extent);i++)
    {
      bit = ( (line/bumps)*key_xlength + i/bumps) % message_length;
      bit_mag[bit] += (*pkey_value * *pkey_value);
        bit_total[bit] += ( *(pimage++) * *(pkey_value++) );
    }
}

  /* fill the message string based on bit_totals */

  for(i=0; i<message_length; i++)
  {
    if(bit_total[i]>0.0)
    {
      message[i]=1;
    }
      else
      {
      message[i]=0;
      }
  }

  for (i = 0; i < message_length; i++)
  {
    // Before normalizing by the magnitudes, be sure we aren't
    // dividing by zero (this happens for an image w/ zero energy.
    if (bit_mag[i] == (float)0.0)
        bit_mag[i] = epsilon;

    bit_total[i] /= (float) sqrt( (double) bit_mag[i] );
  }

  // Compute the "crude metric", an estimate of rms spread of the
// bit level detector's results.  The referenceBitArray is either
// the known message (if it was available to caller) or the
// knewly computed estimate of the message.

*metric = get_crude_metric(referenceBitArray, bit_total, range, message_length);


  delete [] bit_total;
  delete [] bit_mag;
  delete [] key_value;
  delete [] image;
  delete [] wr;
  delete [] wi;

    return;
}
////////////////////////////////////////////////////////////////////////////
// get_read_detail_vector()
//
////////////////////////////////////////////////////////////////////////////
int get_read_detail_vector(
    float *detail_vector,
    unsigned char *data,
    int xdim,
    int row,
    int total_rows,
    int number_channels,
    int start,
    int stop,
    float scale,
    float *image,
    int fftdim
){
    unsigned char *p1;
    float *pdata,*p2;
    int i;
    float base,temp;
    float *pdetail_vector=detail_vector;

    // this function creates a "scaling" vector for the current scan line,
    // based on a crude metric of "local detail"
    if(number_channels == 1){
    }
    else if(number_channels == 3)
    {
        pdata = &image[row*fftdim];
        if(row == 0)p1 = &data[3*row*xdim];
        else p1 = &data[3*(row-1)*xdim];
        if(row == (total_rows-1))p2 = &image[row*fftdim];
        else p2 = &image[(row+1)*fftdim];
        // perform first and last elements outside loop so that an internal if statement is
avoided
        base = (float)*(p1++);base+=(float)*(p1++);base+=(float)*(p1++);
        base+= *(p2++);
        base+= (float)2.0 * *(pdata+1);
        temp = base/(float)4.0 - *(pdata++);
        float denom = (float)(stop-start)/((float)1.0-scale);
        float mult;
        base = (float)fabs( (double)temp );
        if( base > (float)start ){
            if(base > (float)stop)mult = (float)1.0 - scale;
            else mult = (base - (float)start)/denom;
            *(pdetail_vector++) = mult * temp;
        }
        else *(pdetail_vector++) = (float)0.0;
        for(i=1;i<(xdim-1);i++)
        {
            base = (float)*(p1++);base+=(float)*(p1++);base+=(float)*(p1++);
            base+= *(p2++);
            base+= *(pdata+1);
            base+= *(pdata-1);
            temp = base/(float)4.0 - *(pdata++);
            base = (float)fabs( (double)temp );
            if( base > (float)start ){
                if(base > (float)stop)mult = (float)1.0 - scale;
                else mult = (base - (float)start)/denom;
                *(pdetail_vector++) = mult * temp;
            }
            else *(pdetail_vector++) = (float)0.0;
        }
        base = (float)*(p1++);base+=(float)*(p1++);base+=(float)*p1;
        base+= *p2;
        base+= (float)2.0 * *(pdata-1);
        temp = base/(float)4.0 - *pdata;
        base = (float)fabs( (double)temp );
        if( base > (float)start ){
            if(base > (float)stop)mult = (float)1.0 - scale;
            else mult = (base - (float)start)/denom;
            *pdetail_vector = mult * temp;
        }
        else *pdetail_vector = (float)0.0;
    }
    return(1);
}
```

READ.H

```
/////////
// Header file for the Reader core algorithm functions.
/////////
```

```
#ifndef READ_H
#define READ_H

#define SECOND_THRESHOLD  (float) 20.0
#define FIRST_THRESHOLD       (float) 20.0

#define MOV_AV_KERNEL     5

int derivative_threshold(float *data, long length, int number_channels,double maxdiff,
                float filter_cf);
//void float_it(unsigned char *data, float *data_float, long x_extent);
void float_it(unsigned char *data, float *data_float,
          long x_extent, int number_channels);
void remove_mean(float *array, long length);
float get_crude_metric(const unsigned char *actual_message,
                float *bit_total,
                float *range,
                int message_length);


int read_8bit_single_channel_or_color(
  unsigned char *data,            /* input data to be recognized */
  long original_xdim,             /* it's x dimension */
  long original_ydim,             /* it's y dimension */
  long x_offset,                  /* x offset of segment */
  long y_offset,                  /* y offset of segment */
  long x_extent,                  /* x extent of segment */
  long y_extent,                  /* y extent of segment */
  int message_length,             /* length of message in BITS, also length of message string */
  unsigned char *key,             /* original 8 bit random key */
  long key_length,                /* key_length often equal to data_length but not always */
/**unused**/
  char *key_lut,                  /* look up table mapping key value */
  float *luminance_lut,           /* look up table mapping the signature level to luminance*/
  float *detail_lut,              /* look up table mapping the signature level to detail*/

  unsigned char *thumbnail,       /* if available, use pointer, otherwise NULL*/
  unsigned char *original_data,   /* if available, use pointer, otherwise NULL*/

    const unsigned char *referenceBitArray,   // bit array ptr: either the known message or estimate.
  float *metric,                  // we will compute a return a crude metric indicating confidence.
  float *range,
  unsigned char *message,         /* output: either 0 or 1, i.e. inefficient but simple */
  int number_channels,             // generally for B&W==1 vs. color == 3
  int reading_mode,
  int bumps);


void read_8bit_single_channel_OLD_plus_color(
  unsigned char *data,            /* input data to be recognized */
  long original_xdim,             /* it's x dimension */
  long original_ydim,             /* it's y dimension */
  long x_offset,                  /* x offset of segment */
  long y_offset,                  /* y offset of segment */
  long x_extent,                  /* x extent of segment */
  long y_extent,                  /* y extent of segment */
  int message_length,             /* length of message in BITS, also length of message string */
  unsigned char *key,             /* original 8 bit random key */
  long key_length,                /* key_length often equal to data_length but not always */
/**unused**/
  char *key_lut,                  /* look up table mapping key value */
  float *luminance_lut,           /* look up table mapping the signature level to luminance*/
  float *detail_lut,              /* look up table mapping the signature level to luminance*/

  unsigned char *thumbnail,       /* if available, use pointer, otherwise NULL*/
  unsigned char *original_data,   /* if available, use pointer, otherwise NULL*/

    const unsigned char *referenceBitArray,   // bit array ptr: either the known message or estimate.
  float *metric,                  // we will compute a return a crude metric indicating confidence.
  float *range,
  unsigned char *message,         /* output: either 0 or 1, i.e. inefficient but simple */
  int number_channels,
  int bumps);

void read_super(
  unsigned char *data,            /* input data to be recognized */
  long original_xdim,             /* it's x dimension */
  long original_ydim,             /* it's y dimension */
  long x_offset,                  /* x offset of segment */
  long y_offset,                  /* y offset of segment */
  long x_extent,                  /* x extent of segment */
  long y_extent,                  /* y extent of segment */
  int message_length,             /* length of message in BITS, also length of message string */
  unsigned char *key,             /* original 8 bit random key */
  long key_length,                /* key_length often equal to data_length but not always */
/**unused**/
  char *key_lut,                  /* look up table mapping key value */
  float *luminance_lut,           /* look up table mapping the signature level to luminance*/
  float *detail_lut,              /* look up table mapping the signature level to luminance*/
    unsigned char *thumbnail,       /* if available, use pointer, otherwise NULL*/
    unsigned char *original_data,   /* if available, use pointer, otherwise NULL*/

    const unsigned char *referenceBitArray, // bit array ptr: either the known message or
estimate.
    float *metric,                  // we will compute a return a crude metric indicating
confidence.
    float *range,
    unsigned char *message,         /* output: either 0 or 1, i.e. inefficient but simple */
    int number_channels,
    int bumps);

int get_read_detail_vector(
    float *detail_vector,
    unsigned char *data,
    int xdim,
    int row,
    int total_rows,
    int number_channels,
    int start,
    int stop,
    float scale,
    float *image,
    int fftdim
);


#endif  // READ_H
```

READDLG.CPP

```
// readdlg.cpp : implementation file
//

#include "stdafx.h"
#include "signer.h"
#include "readdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// ReadDlg dialog

/////////////////////////////////////////////////////////////////////////
// ReadDlg()
//
// Constructor for the Reader Parameters Dialog object.  A ReadDlg
// object is created to manage a dialog in which the user is able
// to set the parameters used by the Reader and associated core
// algorithms.
/////////////////////////////////////////////////////////////////////////
ReadDlg::ReadDlg(CWnd* pParent /*=NULL*/)
    : CDialog(ReadDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(ReadDlg)
    m_user_key = 0;
    m_msg_length = 0;
    m_gain = (float) 0.0;
    m_bump_size = 0;
    m_detail_lut_scale = 0.0f;
    //}}AFX_DATA_INIT
}

void ReadDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(ReadDlg)
    DDX_Text(pDX, IDC_READ_KEY, m_user_key);
    DDV_MinMaxUInt(pDX, m_user_key, 0, 65535);
    DDX_Text(pDX, IDC_READ_LENGTH, m_msg_length);
    DDV_MinMaxUInt(pDX, m_msg_length, 1, 65535);
    DDX_Text(pDX, IDC_READ_GAIN, m_gain);
    DDV_MinMaxFloat(pDX, m_gain, 1.e-003f, 1.e+006f);
    DDX_Text(pDX, IDC_BUMP_SIZE, m_bump_size);
    DDV_MinMaxInt(pDX, m_bump_size, 1, 256);
    DDX_Text(pDX, IDC_DETAIL_LUT_SCALE, m_detail_lut_scale);
    DDV_MinMaxFloat(pDX, m_detail_lut_scale, 1.e-003f, 1.e+006f);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(ReadDlg, CDialog)
    //{{AFX_MSG_MAP(ReadDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////////
// ReadDlg message handlers

void ReadDlg::OnOK()
{
  CDialog::OnOK();
}
```

READDLG.H

```
// readdlg.h : header file
//

////////////////////////////////////////////////////////////////////////////
// ReadDlg dialog

class ReadDlg : public CDialog
{
// Construction
public:
  ReadDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
  //{{AFX_DATA(ReadDlg)
  enum { IDD = IDD_READ_DIALOG };
  UINT    m_user_key;
  UINT    m_msg_length;
  float   m_gain;
  int     m_bump_size;
  float   m_detail_lut_scale;
  //}}AFX_DATA

// Implementation
protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

  // Generated message map functions
  //{{AFX_MSG(ReadDlg)
  virtual void OnOK();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};
```

RESOURCE.H

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Signer.rc
//
#define IDR_MAINFRAME                   2
#define IDR_DIBTYPE                     3
#define IDD_ABOUTBOX                    100
#define IDC_MESSAGE                     101
#define IDD_PARAMS_DIALOG               101
#define IDC_GAIN                        102
#define IDC_GAIN_LABEL                  103
#define IDD_READ_DIALOG                 103
#define IDC_MESSAGE_LABEL               104
#define IDC_EDIT_GAIN                   106
#define IDC_EDIT_GAMMA                  107
#define IDC_EDIT_KEY                    108
#define IDC_READ_KEY                    110
#define IDC_READ_LENGTH                 111
#define IDC_READ_GAIN                   112
#define IDC_TREE1                       115
#define IDC_BUMP_SIZE                   118
#define IDC_DETAIL_SCALE                120
#define IDC_DETAIL_LUT_SCALE            121
#define ID_EDIT_SETTINGS                32769
#define ID_VIEW_SIGNED                  32770
#define ID_VIEW_UNSIGNED                32771
#define ID_VIEW_SNOW                    32772
#define ID_VIEW_SNOWY_IMAGE             32773
#define ID_VIEW_STATUS                  32774
#define ID_SETTINGS_SIGNER              32775
#define ID_SETTINGS_READER              32776
#define ID_SETTINGS_REGISTRY            32777
#define ID_SETTINGS_AUTOPRINTREPORT     32778
#define ID_SETTINGS_AUTOPRINT           32779
#define ID_OPTIONS_AUTOREAD             32780
#define ID_SETTINGS_AUTOREAD            32781
#define ID_CONTROLS_ALIGN               32782
#define ID_SETTINGS_ALIGN               32783

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        106
#define _APS_NEXT_COMMAND_VALUE         32784
#define _APS_NEXT_CONTROL_VALUE         122
#define _APS_NEXT_SYMED_VALUE           102
#endif
#endif
```

SIGN.CPP

```
////////////////////////////////////////////////////////////////////////////
// FILE: Sign.cpp
//
// DESCRIPTION:
// Core signing functions of the digimarc technology.
// Created July 1995.
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
////////////////////////////////////////////////////////////////////////////

#include "sign.h"
#include <math.h>
#include "stdafx.h"

/* this function loads the scaling factor based on luminance */

int load_luminance_lut( float *luminance_lut, float gamma)   // explicitly written for 8 bit
{
    int i,status=1;

    luminance_lut[0] = (float) 0.; /* don't put any signature energy into zero luminance
(black) */
    for(i=1; i<256; i++)
    {
        luminance_lut[i] =  (float) pow((double)i, (double) gamma);
    }
    return(status);
}


////////////////////////////////////////////////////////////////////////////
// load_key_lut()
//
// This function just assigns mainly 0's, 1's, -1's, 2's and -2's
// to the key values, scaled by the scale_point.
// scale_point is a simple integer between 1 and 127
// about 30 to 50 should be about right for first tests
////////////////////////////////////////////////////////////////////////////
float load_key_lut(char *key_lut, float gain)
{
    int i,base_gain,ifraction;
    float rms,fraction;

    gain /= (float)100.0;
    rms = gain;

    base_gain = (int)gain;
    fraction = gain - (float)base_gain;
    ifraction = (int)( (float) 127.0 * fraction );
    if(ifraction == 0){
        for(i=0;i<128;i++)key_lut[i]=(char)base_gain;
        for(i=0;i<128;i++)key_lut[i+128]=-(char)base_gain;
    }
    else {
        for(i=0;i<(128-ifraction);i++){
            key_lut[i]=(char)base_gain;
            key_lut[i+128]=-(char)base_gain;
        }
        for(i=(128-ifraction);i<128;i++){
            key_lut[i]=(char)(base_gain+1);
            key_lut[i+128]=-(char)(base_gain+1);
        }
    }
    return( rms);
}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//
// The following functions are core algorithms which include
// 1) additional capabilities for signing Color images, and
// 2)
//
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////
// get_detail_vector()
//
////////////////////////////////////////////////////////////////////////
int get_detail_vector(
  float *detail_vector,
  unsigned char *data,
  int xdim,
  int row,
  int total_rows,
  float *detail_lut,
  int number_channels
){
  unsigned char *pdata,*p1,*p2;
  int base,temp,i;
  float *pdetail_vector=detail_vector;

  // this function creates a "scaling" vector for the current scan line,
  // based on a crude metric of "local detail"
  if(number_channels == 1){
      pdata = data;
      if(row == 0)p1 = data;
      else p1 = data - xdim;
      if(row == (total_rows-1))p2 = data;
      else p2 = data + xdim;
      // perform first and last elements outside loop so that an internal if statement is avoided
      base = (int)*(pdata++);
      temp = abs(base- (int)*(p1++));
      temp += abs(base - (int)*(p2++));
      temp += 2*abs(base - (int)*pdata);
      *(pdetail_vector++) = detail_lut[temp];  // make sure it goes up to 1024 elements
      for(i=1;i<(xdim-1);i++){
          base = (int)*(pdata++);
          temp = abs(base- (int)*(p1++));
          temp += abs(base - (int)*(p2++));
          temp += abs(base - (int)*pdata);
          temp += abs(base - (int)*(pdata-2));
          *(pdetail_vector++) = detail_lut[temp];
      }
      base = (int)*pdata;
      temp = abs(base- (int)*p1);
      temp += abs(base - (int)*p2);
      temp += 2*abs(base - (int)*(pdata-1));
      *pdetail_vector = detail_lut[temp];  // make sure it goes up to 1024 elements
  }
  else if(number_channels == 3)
  {
      // use the green channel only just for speed's sake
      pdata = data+1;
      if(row == 0)p1 = data+1;
      else p1 = data+1 - 3*xdim;
      if(row == (total_rows-1))p2 = data+1;
      else p2 = data+1 + 3*xdim;
      // perform first and last elements outside loop so that an internal if statement is avoided
      base = (int)*pdata;pdata+=3;
      temp = abs(base- (int)*p1);p1+=3;
      temp += abs(base - (int)*p2);p2+=3;
      temp += 2*abs(base - (int)*pdata);
      *(pdetail_vector++) = detail_lut[temp];  // make sure it goes up to 1024 elements
      for(i=1;i<(xdim-1);i++)
      {
          base = (int)*pdata;pdata+=3;
          temp = abs(base- (int)*p1);p1+=3;
          temp += abs(base - (int)*p2);p2+=3;
          temp += abs(base - (int)*pdata);
          temp += abs(base - (int)*(pdata-6));
          *(pdetail_vector++) = detail_lut[temp];
      }
      base = (int)*pdata;
      temp = abs(base- (int)*p1);
      temp += abs(base - (int)*p2);
      temp += 2*abs(base - (int)*(pdata-3));
      *pdetail_vector = detail_lut[temp];  // make sure it goes up to 1024 elements
  }
  return(1);
}


////////////////////////////////////////////////////////////////////////
// load_detail_lut()
//
// This function loads the scaling factor based on local detail
////////////////////////////////////////////////////////////////////////
int load_detail_lut( float *detail_lut, float scale)   // explicitly written for 8 bit
{
  int i,status=1;
  float length=(float)(DETAIL_STOP-DETAIL_START);

  scale /= (float)100.0;
  scale*=DETAIL_NORMALIZER;

  for(i=0;i<DETAIL_START;i++)detail_lut[i]=(float)1.0;
  for(i=DETAIL_START; i<DETAIL_STOP; i++)
  {
      detail_lut[i] =  (float)1.0 + scale*((float)(i-DETAIL_START)/length);
  }
  for(i=DETAIL_STOP;i<DETAIL_TOTAL;i++)detail_lut[i]=detail_lut[DETAIL_STOP-1];


  return(status);
}


////////////////////////////////////////////////////////////////////////
// sign_8bit_single_channel_or_color()
//
// written for the march 1996 bump incarnation
////////////////////////////////////////////////////////////////////////
int sign_8bit_single_channel_or_color(
    unsigned char *data,         // input data to be signed
    long data_length,            // it's length
    long xdim,                   // it's x dimension
    long ydim,                   // it's y dimension
    unsigned char *message,      // either 0 or 1, i.e. inefficient but simple
    int message_length,          // length of message in BITS, also length of message string
    unsigned char *key,          // 8 bit random key, uniformly distributed
    long key_length,             // key_length often equal to data_length but not always
*unused*
    char *key_lut,               // look up table mapping key value
    float *luminance_lut,        // look up table mapping the scaling to luminance values
    float *detail_lut,           // look up table mapping the scaling to luminance values
    int signing_mode,            // current options: STANDARD or STRICT_LUMINANCE
    unsigned char *data_out,     // signed output data in same length and format as input
    int number_channels,         // added in late february 1996 to begin work on 3 color 24 bit
color images
    int bumps                    // added in March 1996 to implement bumps
){
    unsigned char *pdata;
    unsigned char *p_out;
    unsigned char *pkey;
    unsigned char *pmessage;
    long i;
    int j,k;
    int lum_change,status=1;
    float ftemp,delta;
    float *detail_vector = new float[xdim];
    float *pdetail_vector,local_gain;
    int key_xlength;

    key_xlength = 1+(xdim-1)/bumps;

    if(number_channels == 1){
        pdata = data;
        p_out = data_out;
        for(i=0;i<ydim;i++){
            // load local detail values for this row
            get_detail_vector(detail_vector,pdata,xdim,i,ydim,detail_lut,number_channels);
            pdetail_vector = detail_vector;
            pkey=&key[(i/bumps)*key_xlength];
            pmessage = &message[((i/bumps)*key_xlength)%message_length];
            for(j=0;j<xdim;j++){
                lum_change = key_lut[(int)*pkey];
                if(lum_change == 0){
                    *(p_out++) = *(pdata++);
                    pdetail_vector++;
                }
                else {
                    local_gain = *(pdetail_vector++) * luminance_lut[*pdata];
                    if( abs(lum_change) > 1 ){   // this is the anti-sparklies check
                        if( local_gain > (float)3.5 ){
                            if(lum_change > 0)lum_change = 1;
                            else lum_change = -1;
                        }
                    }

                    delta = (float)lum_change * local_gain;

                    if( !(*pmessage) )
                        delta = -delta; /* invert current snowy image luminance value ... key
*/

                    ftemp = (float)*(pdata++) + delta;
                    if(ftemp > (float)255.0)*(p_out++) = (unsigned char)255;
                    else if(ftemp<(float)0.0)*(p_out++) = (unsigned char)0;
                    else *(p_out++) = (unsigned char)(ftemp+(float)0.5);
                }

                if( ((j+1)%bumps) == 0 ){
```

```
                pkey++;
                if( (((i/bumps)*key_xlength+j/bumps)%message_length) == (message_length-1) )    /*
 time to restart message */
                {
                    pmessage = message;
                }
                else pmessage++;
            }
        }
    }
  }
  else if(number_channels == 3){
      // data_length is assumed to be the number of pixels, not the number of data bytes
      // RGB packing is assumed, in that order, 3 bytes in a row per pixel: R G B
      if(signing_mode == STANDARD){
          pdata = data;
          p_out = data_out;
          for(i=0;i<ydim;i++){
              // load local detail values for this row
              get_detail_vector(detail_vector,pdata,xdim,i,ydim,detail_lut,number_channels);
              pdetail_vector = detail_vector;
              pkey=&key[(i/bumps)*key_xlength];
              pmessage = &message[((i/bumps)*key_xlength)%message_length];
              for(j=0;j<xdim;j++){
                  lum_change = key_lut[(int)*pkey];
                  if(lum_change == 0){
                      memcpy(p_out,pdata,3*sizeof(unsigned char));
                      pdata+=3;
                      p_out+=3;
                      pdetail_vector++;
                  }
                  else {
                      local_gain = *(pdetail_vector++) * luminance_lut[*(pdata+1)];
                      if( abs(lum_change) > 1 ){   // this is the anti-sparklies check
                          if( local_gain > (float)3.5 ){
                              if(lum_change > 0)lum_change = 1;
                              else lum_change = -1;
                          }
                      }

                      delta = (float)lum_change * local_gain;

                      if( !(*pmessage) )
                          delta = -delta; /* invert current snowy image luminance value ... key */

                      for(k=0;k<3;k++){
                          ftemp = (float)*(pdata++) + delta;
                          if(ftemp > (float)255.0)*(p_out++) = (unsigned char)255;
                          else if(ftemp<(float)0.0)*(p_out++) = (unsigned char)0;
                          else *(p_out++) = (unsigned char)(ftemp+(float)0.5);
                      }
                  }
                  if( ((j+1)%bumps) == 0 ){
                      pkey++;
                      if( (((i/bumps)*key_xlength+j/bumps)%message_length) == (message_length-1) )
/* time to restart message */
                      {
                          pmessage = message;
                      }
                      else pmessage++;
                  }
              }
          }
      }
  }

  return(status);
}
```

SIGN.H

```
//////////////////////////////////////////////////////////////////////////
// FILE: Sign.h
//
// DESCRIPTION:
// Header file for the Signing core algorithms.  Callers of the signing
// functions should include this file.
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//////////////////////////////////////////////////////////////////////////
#ifndef SIGN_H
#define SIGN_H

// These are the possible settings of the "signing_mode" argument
#define STANDARD 0
#define STRICT_LUMINANCE 1
#define LUMINANCE_RED (float)0.31
#define LUMINANCE_GREEN (float)0.59
#define LUMINANCE_BLUE (float)0.11
#define DETAIL_START 20
#define DETAIL_STOP 200
#define DETAIL_TOTAL 1024
#define DETAIL_NORMALIZER (float)7.0


int load_luminance_lut( float *luminance_lut, float gamma );


float load_key_lut( char *key_lut , float gain);


//////////////////////////////////////////////////////////////////////////
// The following function prototypes correspond to the more
// advanced signing algorithms and color image signing capabilities
// added in February 1996.
//////////////////////////////////////////////////////////////////////////

int get_detail_vector(float *detail_vector,
                      unsigned char *data,
                      int xdim,
                      int row,
                      int total_rows,
                      float *luminance_lut,
                      int number_channels);

int load_detail_lut( float *detail_lut, float scale);  // explicitly written for 8 bit

int sign_8bit_single_channel_or_color(
    unsigned char *data,          // input data to be signed
    long data_length,             // it's length
    long xdim,                    // it's x dimension
    long ydim,                    // it's y dimension
    unsigned char *message,       // either 0 or 1, i.e. inefficient but simple
    int message_length,           // length of message in BITS, also length of message string
    unsigned char *key,           // 8 bit random key, uniformly distributed
    long key_length,              // key_length often equal to data_length but not always
*unused*
    char *key_lut,                // look up table mapping key value
    float *luminance_lut,         // look up table mapping the scaling to luminance values
    float *detail_lut,            // look up table mapping the scaling to luminance values
    int signing_mode,             // current options: STANDARD or STRICT_LUMINANCE
    unsigned char *data_out,      // signed output data in same length and format as input
    int number_channels,          // added in late february 1996 to begin work on 3 color 24 bit
color images
    int bumps                     // added in March 1996
);

#endif  // SIGN_H
```

SIGNDOC.CPP

```
//////////////////////////////////////////////////////////////////////////
// FILE: SignDoc.cpp
//
// DESCRIPTION:
// Implementation file for the Document class of the Digimarc Signer.
// This defines the implementation of the document class
// for the Signer.  Under the Microsoft Foundation Class (MFC) architecture,
// the Document/View model is the prefered method.  This header file
// defines our additions to the generic Document class created by the
// Visual C++ wizards.
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "signer.h"
#include <limits.h>

#include "signdoc.h"
#include "signview.h"

#include "coxkey.h"
#include "image.h"
#include "sign.h"
#include "read.h"
#include "align.h"

#include "parmsdlg.h"      // For the Signer Parameters dialog object
#include "readdlg.h"       // For the Reader Parameters dialog object

//#include "AFXPRIV.H"
#include <afxext.h>
#include "mainfrm.h"
```

```
#include <strstrea.h>
#include <fstream.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDibDoc
// Message Map setup.
/////////////////////////////////////////////////////////////////////////////

IMPLEMENT_DYNCREATE(CDibDoc, CDocument)

BEGIN_MESSAGE_MAP(CDibDoc, CDocument)
  //{{AFX_MSG_MAP(CDibDoc)
  ON_COMMAND(ID_SETTINGS_SIGNER, OnSettingsSigner)
  ON_COMMAND(ID_SETTINGS_AUTOPRINT, OnSettingsAutoprint)
  ON_UPDATE_COMMAND_UI(ID_SETTINGS_AUTOPRINT, OnUpdateSettingsAutoprint)
  ON_COMMAND(ID_SETTINGS_READER, OnSettingsReader)
  ON_COMMAND(ID_SETTINGS_AUTOREAD, OnSettingsAutoread)
  ON_UPDATE_COMMAND_UI(ID_SETTINGS_AUTOREAD, OnUpdateSettingsAutoread)
  ON_COMMAND(ID_SETTINGS_ALIGN, OnSettingsAlign)
  ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_AS, OnUpdateFileSaveAs)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDibDoc::CDibDoc()
//
// Constructor for the Signer Document class
/////////////////////////////////////////////////////////////////////////////
CDibDoc::CDibDoc()
{
  //m_hDIB = NULL;
  m_palDIB = NULL;
  m_sizeDoc = CSize(1,1);      // dummy value to make CScrollView happy

  m_hOriginalDIB = NULL;
  m_hSnowyDIB = NULL;
  m_hSignedDIB = NULL;
  m_pRefImage = NULL;
  m_pAlignedImage = NULL;
  m_pAlign = NULL;

  m_pParams = NULL;
  m_pPackedMsg = NULL;

  // Toggles controlled from the "options" menu
  m_autoprint = FALSE;
  m_autoread = ((CDibLookApp *)AfxGetApp())->m_autoread;

  m_state = NO_IMAGE;

  m_filename = "\0";
}

/////////////////////////////////////////////////////////////////////////////
// CDibDoc::~CDibDoc()
//
// Destructor for the Signer Document class.
/////////////////////////////////////////////////////////////////////////////
CDibDoc::~CDibDoc()
{
  if (m_hOriginalDIB != NULL)
  {
      ::GlobalFree((HGLOBAL) m_hOriginalDIB);
      m_hOriginalDIB = NULL;
  }
  if (m_palDIB != NULL)
  {
      delete m_palDIB;
  }

  if (m_hSnowyDIB != NULL)
  {
      ::GlobalFree( (HGLOBAL) m_hSnowyDIB);
      m_hSnowyDIB = NULL;
  }
  if (m_hSignedDIB != NULL)
  {
      ::GlobalFree( (HGLOBAL) m_hSignedDIB);
      m_hSignedDIB = NULL;
  }

  if (m_pPackedMsg != NULL)
      delete m_pPackedMsg;

  if (m_pAlign != NULL)
      delete m_pAlign;
}

/////////////////////////////////////////////////////////////////////////////
// OnNewDocument()
//
/////////////////////////////////////////////////////////////////////////////
BOOL CDibDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// InitDIBData()
//
/////////////////////////////////////////////////////////////////////////////
void CDibDoc::InitDIBData()
{
    if (m_palDIB != NULL)
    {
        delete m_palDIB;
        m_palDIB = NULL;
    }
    if (m_hOriginalDIB == NULL)
    {
        return;
    }
    // Set up document size
    LPSTR lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hOriginalDIB);
    if (::DIBWidth(lpDIB) > INT_MAX ||::DIBHeight(lpDIB) > INT_MAX)
    {
        ::GlobalUnlock((HGLOBAL) m_hOriginalDIB);
        ::GlobalFree((HGLOBAL) m_hOriginalDIB);
        m_hOriginalDIB = NULL;
        MessageBox(NULL, "DIB is too large", NULL,
                         MB_ICONINFORMATION | MB_OK);
        return;
    }
    m_sizeDoc = CSize((int) ::DIBWidth(lpDIB), (int) ::DIBHeight(lpDIB));

    // Save the bits per pixel
    m_BitsPerPixel = ::DIBBitCount(lpDIB);

    ::GlobalUnlock((HGLOBAL) m_hOriginalDIB);

    // Create copy of palette
    m_palDIB = new CPalette;
    if (m_palDIB == NULL)
    {
        // we must be really low on memory
        ::GlobalFree((HGLOBAL) m_hOriginalDIB);
        m_hOriginalDIB = NULL;
        return;
    }
    if (::CreateDIBPalette(m_hOriginalDIB, m_palDIB) == NULL)
    {
        // DIB may not have a palette
        delete m_palDIB;
        m_palDIB = NULL;
        return;
    }
}


CDibView *p_testSignedView;
CCreateContext  newContext;
/////////////////////////////////////////////////////////////////////////////
// OnOpenDocument()
//
/////////////////////////////////////////////////////////////////////////////
BOOL CDibDoc::OnOpenDocument(const char* pszPathName)
{
    extern char *global_cmd_line_args;
    CWinApp *winApp;
    CDibLookApp *myApp;
    CFile file;
    CFileException fe;
    if (!file.Open(pszPathName, CFile::modeRead | CFile::shareDenyWrite, &fe))
    {
        ReportSaveLoadException(pszPathName, &fe,
            FALSE, AFX_IDP_FAILED_TO_OPEN_DOC);
        return FALSE;
    }

    // Get a pointer to the WinApp class object.
    winApp = AfxGetApp();
    myApp = (CDibLookApp *) winApp;
    // TRACE ("Cmd line is: \n\t%s\n", winApp->m_lpCmdLine);
```

```
    // Get pointer to the parameter object.
    m_pParams = myApp->getParams();
    //TRACE ("Gain is: %d\n", m_pParams->GetGain());
    //TRACE ("Filename is: %s\n", m_pParams->GetInputFilename());
    //TRACE ("Message is: %s\n", (const char *) m_pParams->GetMessage());

    DeleteContents();
    BeginWaitCursor();

    // replace calls to Serialize with ReadDIBFile function
    TRY
    {
        m_hOriginalDIB = ::ReadDIBFile(file);
    }
    CATCH (CFileException, eLoad)
    {
        file.Abort(); // will not throw an exception
        EndWaitCursor();
        ReportSaveLoadException(pszPathName, eLoad,
            FALSE, AFX_IDP_FAILED_TO_OPEN_DOC);
        m_hOriginalDIB = NULL;
        return FALSE;
    }
    END_CATCH


    InitDIBData();

    // In debug case, dump out some information about the image.
    // DumpBitmapInfoHeader();

    EndWaitCursor();

    if (m_hOriginalDIB == NULL)
    {
        // may not be DIB format
        MessageBox(NULL, "Couldn't load the \"Original Image\"", NULL,
                    MB_ICONINFORMATION | MB_OK);
        return FALSE;
    }

    // Save the total size needed for the DIB.
    m_dwTotalDIBSize = file.GetLength() - sizeof(BITMAPFILEHEADER);


    SetPathName(pszPathName);
    SetModifiedFlag(FALSE);     // start off with unmodified

    // If we read an 8 or 24 bit image, we're fine; else warn user
    // but we go ahead and display it.
    if (m_BitsPerPixel == 8 || m_BitsPerPixel == 24)
        m_state = IMAGE_LOADED;
    else
    {

        MessageBox(NULL, "The file doesn't contain an 8 or 24 bit image.\n"
                "It will be displayed, but can't be Signed or Read.",
                "Digimarc Signer Warning", MB_ICONINFORMATION | MB_OK);
    }

    return TRUE;
}

//////////////////////////////////////////////////////////////////////////
// OnSaveDocument()
//
//////////////////////////////////////////////////////////////////////////
BOOL CDibDoc::OnSaveDocument(const char* pszPathName)
{
    CFile            file;
    CFileException   fe;
    int              view_type;
    HDIB             hSaveDIB;

    if (!file.Open(pszPathName, CFile::modeCreate |
      CFile::modeReadWrite | CFile::shareExclusive, &fe))
    {
        ReportSaveLoadException(pszPathName, &fe,
            TRUE, AFX_IDP_INVALID_FILENAME);
        return FALSE;
    }

    // replace calls to Serialize with SaveDIB function
    BOOL bSuccess = FALSE;

    // Determine which DIB to save, based on the active window.
    view_type = GetActiveViewType();

    // Set pointer to the DIB of the image which is to be saved.
    if (view_type == ORIGINAL_VIEW)
        hSaveDIB = m_hOriginalDIB;
    else if (view_type == SIGNED_VIEW)
        hSaveDIB = m_hSignedDIB;
    else if (view_type == ALIGNED_VIEW)
        hSaveDIB = m_pAlignedImage->GetHDIB();
    else if (view_type == STATUS_VIEW)
    {
        // This is the unusual case where we are not saving a DIB.
        // Instead, we write out the character strings of the status view.
        file.Close();             // close the binary file, create ofstream instead
        ofstream of(pszPathName);   // Text output file stream
        ostrstream stat_stream;     // For in-memory formatting of the string
        CDibView *stat_view;
        stat_view = GetActiveView();
        stat_view->createStatusStream(stat_stream);
        // Write the status information to the file
        of << stat_stream.str();
        of.close();
        delete stat_stream.str();   // Once we use .str, we have to delete it.
        return TRUE;
    }

    TRY
    {
        BeginWaitCursor();
        bSuccess = ::SaveDIB(hSaveDIB, file);
        file.Close();
    }
    CATCH (CException, eSave)
    {
        file.Abort(); // will not throw an exception
        EndWaitCursor();
        ReportSaveLoadException(pszPathName, eSave,
            TRUE, AFX_IDP_FAILED_TO_SAVE_DOC);
        return FALSE;
    }
    END_CATCH

    EndWaitCursor();
    SetModifiedFlag(FALSE);     // back to unmodified

    if (!bSuccess)
    {
        // may be other-style DIB (load supported but not save)
        //  or other problem in SaveDIB
        MessageBox(NULL, "Couldn't save DIB", NULL,
                    MB_ICONINFORMATION | MB_OK);
    }

    if (m_state == IMAGE_SIGNED_AND_VERIFIED)
        m_state = IMAGE_SIGNED_AND_SAVED;

    // Save the name of the saved file.
    m_filename = pszPathName;

    // If the user switch is set, create a "Status view" (iff it doesn't
    // already exist), and print it.
    if (m_autoprint)
    {
        CDibView *p_status_view;
        p_status_view = (CDibView*) CreateUniqueView(STATUS_VIEW);
        p_status_view->OnFilePrint();
    }
    else
        UpdateAllViews(NULL);   // If status view present, needs update

    return bSuccess;
}

void CDibDoc::ReplaceHDIB(HDIB hDIB)
{
    if (m_hOriginalDIB != NULL)
    {
        ::GlobalFree((HGLOBAL) m_hOriginalDIB);
    }
    m_hOriginalDIB = hDIB;
}


//////////////////////////////////////////////////////////////////////////
// CDibDoc diagnostics

#ifdef _DEBUG
void CDibDoc::AssertValid() const
{
    CDocument::AssertValid();
}
```

```
void CDibDoc::Dump(CDumpContext& dc) const
{
  CDocument::Dump(dc);
}

#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////
// DumpBitmapInfoHeader()
//
// Diagnostic tool which dumps out some information about the DIB's
// header.  Only used for test/debug purposes.
////////////////////////////////////////////////////////////////////////////
void CDibDoc::DumpBitmapInfoHeader() const
{
  int                  i, cxDIB, cyDIB;
  long                 num_pixels, num_colors;
  LPSTR                lpDIB;          // Pointer to BITMAPINFOHEADER
  LPBITMAPINFOHEADER   lpDIBHdr;
  LPBITMAPINFO         lpbmi;

  HDIB hOriginalDIB = GetOriginalHDIB();
  if (hOriginalDIB == NULL)
      return;


  // Lock the  DIB in memory
  lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hOriginalDIB);


  // Get ptr to the  dib header space.
  lpDIBHdr = (LPBITMAPINFOHEADER) lpDIB;

  // get pointer to BITMAPINFO (Win 3.0)
  lpbmi = (LPBITMAPINFO)lpDIB;
    RGBQUAD *bmiColors = lpbmi->bmiColors;

  cxDIB = (int) ::DIBWidth(lpDIB);    // X size of DIB
  cyDIB = (int) ::DIBHeight(lpDIB);   // Y size of DIB

  num_pixels = (long) cxDIB *  cyDIB;

  num_colors = ::DIBNumColors(lpDIB);

  if (lpDIBHdr->biCompression != 0)
  {
      TRACE("Can't cope with compressed image (compression = %d)\n",
                  lpDIBHdr->biCompression);
      ::GlobalUnlock((HGLOBAL) m_hOriginalDIB);
      return;
  }

    TRACE("BITMAPINFOHEADER contents are:\n");
    TRACE("HeaderSize = %ld, width = %d, height = %d, num_pixels = %ld\n",
         lpDIBHdr->biSize, cxDIB, cyDIB, num_pixels);
    TRACE("planes = %d, bitsPerPixel = %d\n",
         lpDIBHdr->biPlanes, lpDIBHdr->biBitCount);
    TRACE("compressionMethod = %d\n", lpDIBHdr->biCompression);
    TRACE("SizeOfBitmap = %d\n", lpDIBHdr->biSizeImage);
    TRACE("num_colors = %d\n", num_colors);
    TRACE("HorzResolution = %ld, VertResolution = %ld\n",
         lpDIBHdr->biXPelsPerMeter, lpDIBHdr->biYPelsPerMeter);
  TRACE("NumColorsUsed = %d NumSigColors = %d\n",
         lpDIBHdr->biClrUsed, lpDIBHdr->biClrImportant);

    // Dump the palette.  This is only for severe debugging situations.
    TRACE("\nThe contents of the palette:\n");
    for (i = 0; i < num_colors; i++)
    {
        TRACE("%d   %2x %2x %2x\n", i,
          (int) bmiColors->rgbRed, (int) bmiColors->rgbGreen,
          (int) bmiColors->rgbBlue);
        bmiColors++;
    }
  // We are now all done w/ the Original DIB.  Unlock it.
  ::GlobalUnlock((HGLOBAL) hOriginalDIB);

}

////////////////////////
// Member function which
// builds a snowy image in place.
//
////////////////////////

typedef char *HPSTR;  // huge pointer to a string NOW OBSOLETE

////////////////////////////////////////////////////////////////////////////

// MakeSnow()
//
// Creates a snowy image, and sets the member variable m_hSnowyDIB, which
// is a DIB handle to the new snowy image DIB.  The snowy image which is
// created is sized based on the parent DIB handle passed in, and it
// has all the same bitmap header and palette stuff.
////////////////////////////////////////////////////////////////////////////
void CDibDoc::MakeSnow(HDIB hParentDIB)
{
    int                   cxDIB, cyDIB;
    long                  num_pixels, num_colors;
    DWORD                 total_size, image_byte;
    LPSTR                 lpDIB, lpSnowyDIB;  // Pointer to BITMAPINFOHEADER
    LPBITMAPINFOHEADER    lpSnowyDIBHdr;
    HPSTR                 hpSnowyDIBBits;
    HPSTR                 src_data, dest_data;       // Huge ptrs for copying the image.

    // HDIB hOriginalDIB = GetOriginalHDIB();
    if (hParentDIB == NULL)
        return;

    // Get the size of the parent DIB
    total_size = GlobalSize( (HGLOBAL) hParentDIB);

    // Create space for the snowy image (on 1st call only).
    if (m_hSnowyDIB == NULL)
    {
        m_hSnowyDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, total_size);
        if (m_hSnowyDIB == 0)
        {
            MessageBox(NULL,
                   "Insufficient memory is availble for the \"snowy image\"",
                   "Digimarc Signer Warning",
                    MB_ICONINFORMATION | MB_OK);
            return;
        }
    }


    // Lock the two DIBs in memory
    lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hParentDIB);
    lpSnowyDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hSnowyDIB);

    src_data = (char *) lpDIB;
    dest_data = (char *) lpSnowyDIB;

    // Copy the BITMAPINFOHEADER, palette, and actual image byte data byte by byte.
    for (image_byte = 0; image_byte < total_size; image_byte++)
    {
        *dest_data++ = *src_data++;
    }

    // For debug: reset the pointers.
    src_data = (char *) lpDIB;
    dest_data = (char *) lpSnowyDIB;
    if (*src_data != *dest_data)
        TRACE("DEBUG: after copy into snowy image, 1st chars aren't equal!\n");

    // We are now all done w/ the Parent DIB.  Unlock it.
    ::GlobalUnlock((HGLOBAL) hParentDIB);


    // Get ptr to the snowy dib header space.
    lpSnowyDIBHdr = (LPBITMAPINFOHEADER) lpSnowyDIB;


    hpSnowyDIBBits = ::FindDIBBits(lpSnowyDIB);


    cxDIB = (int) ::DIBWidth(lpSnowyDIB);    // X size of DIB
    cyDIB = (int) ::DIBHeight(lpSnowyDIB);   // Y size of DIB

    num_pixels = (long) cxDIB *  cyDIB;

    num_colors = ::DIBNumColors(lpSnowyDIB);

    if (lpSnowyDIBHdr->biCompression != 0)
    {
        TRACE("Can't cope with compressed image (compression = %d)\n",
                    lpSnowyDIBHdr->biCompression);
        ::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
        return;
    }


    TRACE("width = %d, height = %d, num_pixels = %ld\n", cxDIB, cyDIB, num_pixels);
    TRACE("num_colors = %d\n", num_colors);

    if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
    {
```

```
    TRACE("At this time, only build snowy image for 8 or 24 bit images\n");
    ::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
    return;
  }

  if (m_BitsPerPixel == 8 || m_BitsPerPixel == 24)
  {

    CoXKey coXKey(m_pParams->GetKey(), (BITMAPINFO *) lpSnowyDIBHdr,
                  hpSnowyDIBBits);
  }

  ::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
}

/////////////////////////////////////////////////////////////////////////////
// Sign()
//
// This is the function which calls upon the core signing algorithms.
//
// WARNING: CURRENTLY THIS FUNCTION ASSUMES THAT WE ALWAYS ARE SIGNING
// THE "ORIGINAL IMAGE" DIB.  THIS MAY BE A BUG.
/////////////////////////////////////////////////////////////////////////////
// First shot at a function which calls the signer core algorithms
void CDibDoc::Sign(void)
{
  long          num_pixels, num_colors;
  DWORD         image_byte;
  HPSTR         src_data, dest_data;        // Huge ptrs for copying the image.
   float        rms;
  int           num_channels;

  HDIB hOriginalDIB = GetOriginalHDIB();
  if (hOriginalDIB == NULL)
    return;

  // Create space for the signed image DIB.
  m_hSignedDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, m_dwTotalDIBSize);
  if (m_hSignedDIB == 0)
  {
    MessageBox(NULL,
               "Insufficient memory is availble for the signed image",
               "Digimarc Signer Warning",
                MB_ICONINFORMATION | MB_OK);
    return;
  }

  // Create Image objects for the images.  Note that this locks them in memory.
  Image snowyImage(m_hSnowyDIB);
  Image unsignedImage(m_hOriginalDIB);

  // This is ugly, but I have to copy the DIB header stuff into the signed DIB
  // before I can create the signedImage object.
  dest_data = (char *) ::GlobalLock( (HGLOBAL) m_hSignedDIB);


  // We want to copy the BITMAPINFO structure from the unsignd to the signed DIB
  src_data = unsignedImage.GetLpDIB();


  // Copy the BITMAPINFOHEADER and palette to the signed DIB space, byte by byte.
  for (image_byte = 0; image_byte < unsignedImage.GetSizeofHeader(); image_byte++)
  {
    *dest_data++ = *src_data++;
  }

  ::GlobalUnlock( (HGLOBAL) m_hSignedDIB);

  // Now create the signedImage object, which will lock the DIB in memory again.
  Image signedImage(m_hSignedDIB);

  // For each, create a "byte-wise" packed data array from the  DIB 4-byte packing
  snowyImage.MakePackedData(FORCE_TO_1_CHANNEL);  // snowy image always 1 chan
  unsignedImage.MakePackedData();
  signedImage.MakePackedData();


  num_pixels = (long) unsignedImage.GetXDim() * unsignedImage.GetYDim();

  num_colors = unsignedImage.GetNumColors();


  if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
  {
    TRACE("At this time, only sign 8 and 24 bit images\n");
    return;
  }

  // Create and load the luminance scaling look up table.
```

```
  float *luminance_lut = new float[256];
  ::load_luminance_lut(luminance_lut, m_pParams->GetGamma());

  // Create and load the key look up table.
  char *key_lut = new char[256];
  rms = ::load_key_lut(key_lut, m_pParams->GetGain());

  long data_length = unsignedImage.GetXDim() * unsignedImage.GetYDim();

  // Create a packed msg (will be a user input in future).
  if (m_pPackedMsg != NULL)
    delete m_pPackedMsg;
  m_pPackedMsg = new PackedMsg( (const char *) m_pParams->GetMessage());


  // Set up some arguments and call the core signer.
  int     x_dim = unsignedImage.GetXDim();
  int     y_dim = unsignedImage.GetYDim();

  if (unsignedImage.GetBitsPerPixel() == 8)
    num_channels = 1;
  else if (unsignedImage.GetBitsPerPixel() == 24)
    num_channels = 3;

  // const float lut_scale = (float)1.0;  // Later this will be user controlled.
  float *detail_lut = new float[DETAIL_TOTAL];
  ::load_detail_lut(detail_lut, m_pParams->GetLutScale());

  ::sign_8bit_single_channel_or_color(unsignedImage.GetPackedData(),
                                      data_length,
                                      x_dim,
                                      y_dim,
                                      m_pPackedMsg->getMsgBitArray(),
                                      m_pPackedMsg->getMsgBitArrayLength(),
                                      snowyImage.GetPackedData(),
                                      data_length,
                                      key_lut,
                                      luminance_lut,
                                      detail_lut,
                                      STANDARD,
                                      signedImage.GetPackedData(),
                                      num_channels,
                                      m_pParams->GetBumpSize());

  delete [] detail_lut;

  // Set the timestamp indicating when we signed this puppy.
  m_pParams->UpdateSignTime();


  delete [] luminance_lut;
  delete [] key_lut;

  // Now unpack the data in the Image object, back into the standard DIB format
  signedImage.UnpackData();

}

/////////////////////////////////////////////////////////////////////////////
// Read()
//
// The read function is the interface to the core recognition algorithm.
// It sets up the necessary data structures needed by the core routine
// and makes the call.
/////////////////////////////////////////////////////////////////////////////
void CDibDoc::Read(HDIB hSignedDIB, BOOL use_super_reader)
{
    long     num_pixels, num_colors;
    int      num_channels;
    int      reading_mode;

    // Create Image objects for the images.  Note that this locks them in memory.
    Image snowyImage(m_hSnowyDIB);

    Image signedImage(hSignedDIB);

    // Create a "byte-wise" packed data array from the DIB 4-byte packing
    signedImage.MakePackedData();
    snowyImage.MakePackedData(FORCE_TO_1_CHANNEL);//Snowy images always 1 ch.
    // unsignedImage.MakePackedData();


    num_pixels = (long) signedImage.GetXDim() * signedImage.GetYDim();

    num_colors = signedImage.GetNumColors();


    if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
    {
```

```
    TRACE("At this time, only recognize 8 and 24 bit images\n");
    return;
}

// Create and load the luminance scaling look up table.
float *luminance_lut = new float[256];
::load_luminance_lut(luminance_lut, m_pParams->GetGamma());

// Create and load the key look up table.
char *key_lut = new char[256];
::load_key_lut(key_lut, m_pParams->GetGain());

// Create and load the detail look up table.
float *detail_lut = new float[DETAIL_TOTAL];
//const float lut_scale = (float)1.0;   // Later this will be user controlled.
::load_detail_lut(detail_lut, m_pParams->GetLutScale());

// Determine which bit array to use for the reader's "crude metric"
// computation.  If we have just signed this image, then use the
// true message bit array.  Otherwise, we are trying to read
// without knowing the the true message, and use the estimated
// message for computation of the metric.
  unsigned char *referenceBitArray;
if (m_state == IMAGE_SIGNED || m_state == IMAGE_SIGNED_AND_VERIFIED ||
    m_state == IMAGE_SIGNED_AND_SAVED)
    referenceBitArray = m_pPackedMsg->getMsgBitArray();
else
    referenceBitArray = m_pPackedMsg->getReaderBitArray();

long data_length = signedImage.GetXDim() * signedImage.GetYDim();
long    x_offset = 0;
long    y_offset = 0;
int     x_dim = signedImage.GetXDim();
int     y_dim = signedImage.GetYDim();


if (signedImage.GetBitsPerPixel() == 8)
    num_channels = 1;
else if (signedImage.GetBitsPerPixel() == 24)
    num_channels = 3;

// See if we should use the super reader.
if (use_super_reader)
    reading_mode = 1;
else
    reading_mode = 0;
// Call the core recognizer
::read_8bit_single_channel_or_color(
                            signedImage.GetPackedData(),
                            x_dim,
                            y_dim,
                            x_offset,
                            y_offset,
                            x_dim,      // segment is full image.
                            y_dim,
                            m_pPackedMsg->getMsgBitArrayLength(),
                            snowyImage.GetPackedData(),
                            data_length,
                            key_lut,
                            luminance_lut,
                            detail_lut,
                            NULL,                           // No thumbnail at this time
                            //unsignedImage.GetPackedData(),
                            NULL,       // Don't pass original data now
                             (const unsigned char *) referenceBitArray,
                            &m_crude_metric,
                            &m_range,
                            m_pPackedMsg->getReaderBitArray(),
                            num_channels,
                            reading_mode,
                            m_pParams->GetBumpSize());


// Convert the recovered message bits back to an ASCII string.
m_pPackedMsg->BitsToString();

TRACE ("The recognizer detected the following string: %s\n",
        m_pPackedMsg->getRecoveredAsciiMsg());


delete [] luminance_lut;
delete [] key_lut;
delete [] detail_lut;

}
/////////////////////////////////////////////////////////////////////////////
// CDibDoc commands
```

```
/////////////////////////////////////////////////////////////////////////////
// OnSettingsSigner()
//
// This function is invoked when the user selects the Settings-->
// Signer Controls... menu item.  It creates a Signer parameters
// dialog object and presents it to the user as a modal dialog.
// If the user presses OK, we then gather the new parameter values
// and use them to sign the image.  Finally, a new view and window
// are created to display the signed image, if no such view already
// exists.
/////////////////////////////////////////////////////////////////////////////
void CDibDoc::OnSettingsSigner()
{
    ParmsDlg    dlg;
    CRect       rect;
    unsigned    old_key;
    BOOL        new_user_key = FALSE;


    // Check to see if we are in a legal state for signing.
    if (m_state == NO_IMAGE)
    {
        MessageBox(NULL,
                "An 8 or 24 bit image must be loaded before using the Signer.",
                "Digimarc Signer Warning",
                 MB_ICONINFORMATION | MB_OK);
        return;
    }
    // int  scroll_pos

    // Initialize the dialog data
    dlg.m_message = m_pParams->GetMessage();
    dlg.m_gain_from_edit_box = m_pParams->GetGain();
    // dlg.m_gamma = m_pParams->GetGamma(); gamma no longer user cntrl
    dlg.m_key = m_pParams->GetKey();
    old_key = m_pParams->GetKey();
    dlg.m_bump_size = m_pParams->GetBumpSize();
    dlg.m_detail_lut_scale = m_pParams->GetLutScale();

    // Get the coordinates for the scroll bar object window.
    // dlg.m_gain.GetWindowRect(&rect);

    // Try to "create" the scroll bar.
    // dlg.m_gain.Create(WS_CHILD, CRect(10, 50, 200, 20), &dlg, IDC_GAIN);

    // Invoke the dialog box
    if (dlg.DoModal() == IDOK)
    {
        // retrieve the dialog data
        m_pParams->SetMessage(dlg.m_message);

        if (dlg.m_key != old_key)
        {
            m_pParams->SetKey(dlg.m_key);
            new_user_key = TRUE;
        }

        m_pParams->SetGain(dlg.m_gain_from_edit_box);

        m_pParams->SetBumpSize(dlg.m_bump_size);

        m_pParams->SetLutScale(dlg.m_detail_lut_scale);

        // m_pParams->SetGamma(dlg.m_gamma);// gamma no longer user cntrl

        // scroll_pos = dlg.m_gain.GetScrollPos();

        // TRACE("Scrollbar position: %d\n", scroll_pos);

        // This is going to take awhile
        BeginWaitCursor();


        // NOTE: AT THIS POINT SHOULD DETERMINE WHAT IMAGE IS IN THE
        // ACTIVE VIEW, AND IF IT CONTAINS A BITMAP SIGN THAT IMAGE.
        // SEE OnSettingsReader(), which uses the correct logic.
        // Then, call MakeSnow(hImageToSignDIB) and Sign(hImageToSignDIB)

        // If the user seed has changed, or if we haven't yet created
        // a coextensive key, create a snowy image.
        if (new_user_key || m_hSnowyDIB == NULL)
            MakeSnow(m_hOriginalDIB);

        // Use the new settings, and sign the image.
        Sign();

        m_state = IMAGE_SIGNED;

        if (((CDibLookApp *)AfxGetApp())->m_autoread)
```

```
    {
        // Run the reader again to see if we recover message.
        Read(m_hSignedDIB, FALSE);

        m_state = IMAGE_SIGNED_AND_VERIFIED;
    }

    // Now see if a "signed image" view exists.  If not, create it.
    CreateUniqueView(SIGNED_VIEW);

    // Now see if a "status image" view exists.  If not, create it.
    CDibView *p_statusView;
    p_statusView = (CDibView *) CreateUniqueView(STATUS_VIEW);

    EndWaitCursor();

    // Refresh all of the views  (Don't actually need to refresh Original one)
    p_statusView->DoResize();
    UpdateAllViews(NULL);

    // Some debug stuff related to checksums.
    TRACE("Signer checksum: %x\n", (int) m_pPackedMsg->GetSignerChecksum());
    TRACE("Read checksum: %x\n", (int) m_pPackedMsg->GetReaderChecksum());
    TRACE("Reader computed checksum: %x\n",
            (int) m_pPackedMsg->GetComputedReaderChecksum());
  }
}

//////////////////////////////////////////////////////////////////////
// CreateUniqueView()
//
// This function creates a new view of the indicated type, if and
// only if one does not already exist.  It returns a pointer to
// the new view, if a new one is created, or a pointer to the
// pre-existing view of the specified type if one already exists.
// The "view_type" argument is one of the view types from SignView.h,
// i.e. SIGNED_VIEW, ORIGINAL_VIEW, STATUS_VIEW.
//////////////////////////////////////////////////////////////////////
CView* CDibDoc::CreateUniqueView(int view_type)
{
   BOOL view_found = FALSE;
  POSITION pos = GetFirstViewPosition();
  CView* pView;
  while (pos != NULL)
  {
      pView = GetNextView( pos );

      // If we find it, we return the pointer and we're done.
      if ( ((CDibView*)pView)->GetViewType() == view_type)
          return pView;
  }

  // The desired type of view doesn't exist, so we create it.

    CMainFrame *mainFrame = (CMainFrame *) AfxGetApp()->m_pMainWnd;
    mainFrame->MyOnWindowNew();


  // Now find the newly created view (last in list) and set its type.
  pos = GetFirstViewPosition();
  while (pos != NULL)
      pView = GetNextView(pos);

  ((CDibView*)pView)->SetViewType(view_type);

  return(pView);
}


//////////////////////////////////////////////////////////////////////
// ChangeViewType()
//
// This function finds the view of the "old_type", and changes its
// type to "new_type".  If successful, it returns a pointer to
// the newly changed view.  If not, returns NULL.
// The "view_type" arguments are from the view types in SignView.h,
// i.e. SIGNED_VIEW, ORIGINAL_VIEW, STATUS_VIEW, ALIGNED_VIEW, ...
//////////////////////////////////////////////////////////////////////
CView* CDibDoc::ChangeViewType(int old_type, int new_type)
{
   BOOL view_found = FALSE;
  POSITION pos = GetFirstViewPosition();
  CView* pView;
  while (pos != NULL)
  {
      pView = GetNextView( pos );

      // If we find it, change its type we return the pointer and we're done.
      if ( ((CDibView*)pView)->GetViewType() == old_type)
      {
            ((CDibView*)pView)->SetViewType(new_type);
            return pView;
      }
  }
      // We get here only if we failed to find a view of "old_type"
      return NULL;
}

///////////////////////////////////////////////////////////////////
// OnSettingsAutoprint()
//
// When the user toggles the "Auto-print Report" item in
// the Options menu, this function is invoked.  It simply
// toggles the corresponding member variable.
///////////////////////////////////////////////////////////////////
void CDibDoc::OnSettingsAutoprint()
{
    if (m_autoprint == TRUE)
        m_autoprint = FALSE;
    else
        m_autoprint = TRUE;
}

///////////////////////////////////////////////////////////////////
// OnUpdateSettingsAutoprint()
//
// The framework calls this function whenever it is about
// to dislay the pulldown menu containing the Autoprint
// Report option.  Based on our internal state variable
// m_autoprint, we set or clear the check mark next to
// the menu item using the pCmdUI->SetCheck() function.
///////////////////////////////////////////////////////////////////
void CDibDoc::OnUpdateSettingsAutoprint(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if (m_autoprint == TRUE)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);

}

///////////////////////////////////////////////////////////////////
// OnSettingsReader()
//
// Invoked when the user selects the Controls-->Reader...
// menu option.  Presents a ReadParmsDlg dialog object, and
// deals with the operators inputs.  On OK, the Read() function
// is called to use the current parameters and run the recog-
// nition core algorithms to try to detect an embedded
// digimarc message.
///////////////////////////////////////////////////////////////////
void CDibDoc::OnSettingsReader()
{
    ReadDlg      dlg;
    CRect        rect;
    unsigned     old_key;
    BOOL         new_user_key = FALSE;
    int          view_type;
    HDIB         hImageToReadDIB;

    // Check to see if we are in a legal state for reading.
    if (m_state == NO_IMAGE)
    {
        MessageBox(NULL,
                   "An 8 or 24 bit image must be loaded before using the Reader.",
                   "Digimarc Signer Warning",
                   MB_ICONINFORMATION | MB_OK);
        return;
    }

    // Determine the type of the active window
    view_type = GetActiveViewType();

    // If active window is not acceptable for reading, warn user & return
    if (view_type != ORIGINAL_VIEW &&
        view_type != SIGNED_VIEW &&
        view_type != ALIGNED_VIEW)
    {
        MessageBox(NULL,
                    "The active window must contain an image to be read.",
                    "Warning",
                    MB_ICONINFORMATION | MB_OK);
        return;
    }

    // Set pointer to the image which is to be read.
    if (view_type == ORIGINAL_VIEW)
```

```
        hImageToReadDIB = m_hOriginalDIB;
    else if (view_type == SIGNED_VIEW)
        hImageToReadDIB = m_hSignedDIB;
    else if (view_type == ALIGNED_VIEW)
        hImageToReadDIB = m_pAlignedImage->GetHDIB();
    else
    {
        MessageBox(NULL, "Bug in OnSettingsReader!", "Error", MB_OK);
        return;
    }

    // Initialize the dialog data
    dlg.m_user_key = m_pParams->GetKey();
    old_key = m_pParams->GetKey();
    dlg.m_msg_length = m_pParams->GetMessage().GetLength();
    dlg.m_gain = m_pParams->GetGain();
    dlg.m_bump_size = m_pParams->GetBumpSize();
    dlg.m_detail_lut_scale = m_pParams->GetLutScale();
    // dlg.m_use_super_reader = m_pParams->GetSuperReaderFlag();


    // Invoke the dialog box
    if (dlg.DoModal() == IDOK)
    {
        m_pParams->SetGain(dlg.m_gain);
        m_pParams->SetBumpSize(dlg.m_bump_size);
        m_pParams->SetLutScale(dlg.m_detail_lut_scale);
        // m_pParams->SetSuperReaderFlag(dlg.m_use_super_reader);

        // If signer has not yet been used, or length changes, need a msg.
        if (m_pParams->GetMessage().GetLength() != (int) dlg.m_msg_length)
        {
            // Create a dummy msg of all x's.
            CString dummy_msg = CString('x', dlg.m_msg_length);
            m_pParams->SetMessage(dummy_msg);
        }

        // Create a PackedMsg object w/ our dummy msg.
        if (m_pPackedMsg != NULL)
            delete m_pPackedMsg;
        m_pPackedMsg = new PackedMsg( (const char *) m_pParams->GetMessage());

        if (dlg.m_user_key != old_key)
        {
            m_pParams->SetKey(dlg.m_user_key);
            new_user_key = TRUE;
        }

        // This is going to take awhile
        BeginWaitCursor();

        // If the user seed has changed, or if we haven't yet created
        // a coextensive key, create a snowy image.
        if (new_user_key || m_hSnowyDIB == NULL)
            MakeSnow(hImageToReadDIB);

        // Run the reader and attempt to recover message, and compute metrics.
        Read(hImageToReadDIB, m_pParams->GetSuperReaderFlag());

        // Make the state transition: depends on which image was read.
        if (view_type == ORIGINAL_VIEW || view_type == ALIGNED_VIEW)
            m_state = SUSPECT_READ;
        else if (view_type == SIGNED_VIEW)
        {
            if (m_state != IMAGE_SIGNED_AND_SAVED)
                m_state = IMAGE_SIGNED_AND_VERIFIED;
        }

        // KLUDGE for debug.  Need the signer timestamp set.
//      WHY? 11/24
//          m_pParams->UpdateSignTime();

        // Now see if a "status image" view exists.  If not, create it.
        CDibView *p_statusView;
        p_statusView = (CDibView *) CreateUniqueView(STATUS_VIEW);

        EndWaitCursor();

        // Refresh all of the views  (Don't actually need to refresh Original one)
        p_statusView->DoResize();
        UpdateAllViews(NULL);

        // See if the checksum read and the checksum computed from the
        // read message string agree.  If not, warn user.
        if (m_pPackedMsg->GetReaderChecksum() !=
            m_pPackedMsg->GetComputedReaderChecksum())
        {
            MessageBox(NULL,
                "The embedded checksum didn't match the computed checksum.",
                "Warning", MB_OK);
        }
    }
}

//////////////////////////////////////////////////////////////
// GetActiveViewType()
//
// Find the active view, determine its type, and return
// it to the caller.  The type is one of those listed
// in the DibView.h file.
//////////////////////////////////////////////////////////////
int CDibDoc::GetActiveViewType(void)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView* pView;
    while (pos != NULL)
    {
        pView = GetNextView( pos );

        // If we find it, we return the pointer and we're done.
        if ( ((CDibView*)pView)->IsViewActive() == TRUE)
            return ((CDibView*)pView)->GetViewType();
    }

    // We can get here when other apps are running and Windows sends message
    // resulting in CDibDoc::OnUpdateFileSaveAs() being called.
    // MessageBox(NULL, "Error in GetActiveViewType!", "Error", MB_OK);
    return(UNKNOWN_VIEW);
}


//////////////////////////////////////////////////////////////////////
// GetActvieView()
//
// Return a pointer to the active view (i.e., a CDibView* ), or NULL
// if something goes wrong.
//////////////////////////////////////////////////////////////////////
CDibView * CDibDoc::GetActiveView(void)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView* pView;
    while (pos != NULL)
    {
        pView = GetNextView( pos );

        // If we find it, we return the pointer and we're done.
        if ( ((CDibView*)pView)->IsViewActive() == TRUE)
            return (CDibView*)pView;
    }

    // We can get here when other apps are running and Windows sends message
    // resulting in CDibDoc::OnUpdateFileSaveAs() being called.
    // MessageBox(NULL, "Error in GetActiveViewType!", "Error", MB_OK);
    return(NULL);
}


//////////////////////////////////////////////////////////////
// OnSettingsAutoread()
//
// When the user toggles the "Auto-read after Signing" item in
// the Options menu, this function is invoked.  It simply
// toggles the corresponding member variable.
//
// We currently also toggle the application level variable,
// so that the settings are global to all docs.
//////////////////////////////////////////////////////////////
void CDibDoc::OnSettingsAutoread()
{
    if (m_autoread == TRUE)
    {
        m_autoread = FALSE;
        ((CDibLookApp *)AfxGetApp())->m_autoread = FALSE;
    }
    else
    {
        m_autoread = TRUE;
        ((CDibLookApp *)AfxGetApp())->m_autoread = TRUE;
    }
}

//////////////////////////////////////////////////////////////////////
// OnUpdateSettingsAutoread()
//
// The framework calls this function whenever it is about
// to dislay the pulldown menu containing the Autoread
// option.  Based on our internal state variable
```

```
// m_autoread, we set or clear the check mark next to
// the menu item using the pCmdUI->SetCheck() function.
//////////////////////////////////////////////////////////////////////////////
void CDibDoc::OnUpdateSettingsAutoread(CCmdUI* pCmdUI)
{
  // Set or clear the check mark in the menu
    if (((CDibLookApp *)AfxGetApp())->m_autoread == TRUE)
      pCmdUI->SetCheck(TRUE);
  else
      pCmdUI->SetCheck(FALSE);

}

//////////////////////////////////////////////////////////////////////////////
// OnSettingsAlign()
//
// This function is called when the user selects the "Align" menu option.
// A CFileDialog object is created and used in order for the operator
// to specify the name of the "Reference Image" (a signed or unsigned
// original image used as the template).
//////////////////////////////////////////////////////////////////////////////
void CDibDoc::OnSettingsAlign()
{
  CString refname;
  BOOL success_flag;

  // Create a filter for the types of files the file dialog will offer
  char szFilter[] =
      "Windows Bit Map Files (*.bmp)|*.bmp|Device Independent Bitmaps (*.dib)|*.dib|"
      "All Files (*.*)|*.*||";

  // Construct a file dialog
  CFileDialog     fileDlg(TRUE,       // its a file open (not save) dialog
                  "*.BMP",
                  NULL,
                  OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
                  szFilter);

  // Over-ride the default title in the file dialog window
  fileDlg.m_ofn.lpstrTitle = "Select a template file to be used for alignment";

  // Display the file dialog
  if (fileDlg.DoModal() == IDOK)
  {
      // Get the name of the reference image file.
      refname = fileDlg.GetPathName();

      BeginWaitCursor();

      // Create an Image object for the reference image.
      // (If one already exists, delete it first).
      if (m_pRefImage != NULL)
          delete m_pRefImage;
      m_pRefImage = new Image(refname);

      if (m_pRefImage->GetFileOK == FALSE) // bail out if something went wrong
          return;

      // Display the reference image
//        CreateUniqueView(REF_VIEW);

//        UpdateAllViews(NULL);

      TRACE("Call the Align() function (this is a test of trace output.)\n");

      // Do the actual alignment and change update the state description.
      success_flag = Align_it();

      if (success_flag)
      {
          m_state = SUSPECT_ALIGNED;

          // Now, the template image object has had its packed data array replaced
          // by the aligned, co-extensive image.  Need to move this packed data
          // into the DIB array for display (and possible file saving) purposes.
          m_pRefImage->UnpackData();

          // We now call the image the Aligned image, not reference
          m_pAlignedImage = m_pRefImage;
          m_pRefImage = NULL;

          CreateUniqueView(ALIGNED_VIEW);

          // Create a status view, if it doesn't already exist.
          CDibView *p_statusView;
          p_statusView = (CDibView *) CreateUniqueView(STATUS_VIEW);

          p_statusView->DoResize();

          UpdateAllViews(NULL);
      }

      EndWaitCursor();
  }
}

//////////////////////////////////////////////////////////////////////////////
// Align_it()
//
// This function is responsible for carrying out the alignment operation,
// by calling upon Geoff's core algorithms.  It is assumed that on entry
//      1) m_hOriginalDIB is DIB of the suspect image, already loaded.
//      2) m_pRefImage points to a Image object with the template (or
//      reference) image.
//
//////////////////////////////////////////////////////////////////////////////
BOOL CDibDoc::Align_it(void)
{
    int num_channels;

    // Create an image object for the supect image.
    Image suspectImage(m_hOriginalDIB);


    // Currently we require that the reference and suspect are of same type
    // (i.e., both color or B&W).
    if (suspectImage.GetBitsPerPixel() != m_pRefImage->GetBitsPerPixel())
    {
        MessageBox(NULL,
                   "The suspect and reference images must both be color or B&W",
                   "Warning",
                   MB_ICONINFORMATION | MB_OK);
        return(FALSE);
    }

    // Construct Align object.
    if (m_pAlign != NULL)
        delete m_pAlign;

    m_pAlign = new Align;

    // Create the "byte-wise" packed data arrays from the DIB 4-byte packing
    suspectImage.MakePackedData();
    m_pRefImage->MakePackedData();

    if (suspectImage.GetBitsPerPixel() == 8)
        num_channels = 1;               // B&W image
    else if (suspectImage.GetBitsPerPixel() == 24)
        num_channels = 3;               // Color image


    // Call the core algorithm to do the alignment.
    m_pAlign->direct_registration(m_pRefImage->GetPackedData(),
                                  m_pRefImage->GetXDim(),
                                  m_pRefImage->GetYDim(),
                                  suspectImage.GetPackedData(),
                                  suspectImage.GetXDim(),
                                  suspectImage.GetYDim(),
                                  num_channels);
    return(TRUE);
}


//////////////////////////////////////////////////////////////////////////////
// OnUpdateFileSaveAs()
//
// When the File pulldown menu is selected, this function is called
// upon to determine whether the "Save As..." menu item should be
// enabled.  It determines the type of the current view, and if it
// is of a type for which we currently allow file saves, the menu
// item is enabled.
//////////////////////////////////////////////////////////////////////////////
void CDibDoc::OnUpdateFileSaveAs(CCmdUI* pCmdUI)
{
    int view_type;

    // Determine the type of the current view.
    view_type = GetActiveViewType();

    // If the active view contains an image, we know how to save it.
    if (view_type == ORIGINAL_VIEW ||
        view_type == SIGNED_VIEW ||
        view_type == ALIGNED_VIEW ||
        view_type == STATUS_VIEW)
    {
        pCmdUI->Enable(TRUE);
    }
    else
```

```
    pCmdUI->Enable(FALSE);
}
```

SIGNDOC.H

```
//////////////////////////////////////////////////////////////////////////
// FILE: SignDoc.h
//
// DESCRIPTION:
// Interface file for the CDibDoc class.  This defines the document class
// for the Signer.  Under the Microsoft Foundation Class (MFC) architecture,
// the Document/View model is the prefered method.  This header file
// defines our additions to the generic Document class created by the
// Visual C++ wizards.
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//////////////////////////////////////////////////////////////////////////

#include "dibapi.h"
#include "packmsg.h"
#include "params.h"
#include "Image.h"
#include "Align.h"

//#include "signview.h"

// Define the possible states...
#define NO_IMAGE                    0
#define IMAGE_LOADED                1
#define IMAGE_SIGNED                2
#define   IMAGE_SIGNED_AND_VERIFIED    3
#define SUSPECT_READ                4
#define IMAGE_SIGNED_AND_SAVED         5
#define SUSPECT_ALIGNED                6


#define FORCE_TO_1_CHANNEL TRUE       // For clarity when packing rgb images to 1 chan.


class CDibView;

class CDibDoc : public CDocument
{
protected: // create from serialization only
  CDibDoc();
  DECLARE_DYNCREATE(CDibDoc)

// Attributes
public:
  // HDIB GetHDIB() const
  //  { return m_hDIB; }

  HDIB GetSignedHDIB() const
      { return m_hSignedDIB; }
  HDIB GetOriginalHDIB() const
      { return m_hOriginalDIB; }
  HDIB GetSnowyHDIB() const
      { return m_hSnowyDIB; }
  HDIB GetRefHDIB() const
      { return m_pRefImage->GetHDIB(); }
  HDIB GetAlignedHDIB() const
      { return m_pAlignedImage->GetHDIB(); }

  CPalette* GetDocPalette() const
      { return m_palDIB; }
  CSize GetDocSize() const
      { return m_sizeDoc; }

  PackedMsg *GetPackedMsg() const
      { return m_pPackedMsg; }

  SignerParams *GetSignerParams() const
      { return m_pParams; }

  int GetState() const {return m_state;}

  const CString& GetFilename() const {return m_filename;}

  float GetMetric() const {return m_crude_metric;}
  float GetRange() const {return m_range;}

  // Accessors so view objects can get alignment results.
  const AlignStatus GetAlignStatus(void) const {return m_pAlign->GetAlignStatus();}


// Operations
public:
  void ReplaceHDIB(HDIB hDIB);
    void InitDIBData();

// Implementation
protected:
    virtual ~CDibDoc();

    virtual BOOL OnSaveDocument(const char* pszPathName);
    virtual BOOL OnOpenDocument(const char* pszPathName);

    //void OnEditSettings();

private:
    void MangleDIB(void);
    void CDibDoc::DumpBitmapInfoHeader() const;
    void MakeSnow(HDIB hParentDIB);
    void Sign(void);
    void Read(HDIB hSignedDIB, BOOL use_super_reader);
    BOOL Align_it(void);
    CView* CreateUniqueView(int view_type);
    CView* ChangeViewType(int old_type, int new_type);
    int GetActiveViewType(void);

    CDibView *GetActiveView(void);

    int m_state;
    CString m_filename;

    float m_crude_metric;
    float m_range;

    Image *m_pRefImage;
    Image *m_pAlignedImage;

    Align *m_pAlign;

protected:
    // HDIB m_hDIB;          // Obsolete
    CPalette* m_palDIB;
    CSize m_sizeDoc;
    int m_BitsPerPixel;

    CView *m_pSignedView;

    // Ptr to the initially loaded image, unmodified by signing.
    HDIB m_hOriginalDIB;

    // Add additional DIB handles for the snowy image and signed image.
    HDIB m_hSnowyDIB;
    HDIB m_hSignedDIB;



    // Need to know total space needed for these guys.
    DWORD m_dwTotalDIBSize;

    // Pointer to parameters object.
    SignerParams *m_pParams;

    PackedMsg *m_pPackedMsg;

    BOOL m_autoprint;
    BOOL m_autoread;


#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
    virtual BOOL    OnNewDocument();

// Generated message map functions
protected:
    //{{AFX_MSG(CDibDoc)
    afx_msg void OnSettingsSigner();
    afx_msg void OnSettingsAutoprint();
    afx_msg void OnUpdateSettingsAutoprint(CCmdUI* pCmdUI);
    afx_msg void OnSettingsReader();
    afx_msg void OnSettingsAutoread();
    afx_msg void OnUpdateSettingsAutoread(CCmdUI* pCmdUI);
    afx_msg void OnSettingsAlign();
    afx_msg void OnUpdateFileSaveAs(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//////////////////////////////////////////////////////////////////////////
```

SIGNER.CPP

```
// signer.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "signer.h"

#include "mainfrm.h"
#include "signdoc.h"
#include "signview.h"

#include "mychildw.h"

// #include "AFXPRIV.H"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

char *global_cmd_line_args;
/////////////////////////////////////////////////////////////////////////////
// CDibLookApp

BEGIN_MESSAGE_MAP(CDibLookApp, CWinApp)
  //{{AFX_MSG_MAP(CDibLookApp)
  ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
  //}}AFX_MSG_MAP
  // Standard file based document commands
  ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
  ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
  // Standard print setup command
  ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDibLookApp construction
// Place all significant initialization in InitInstance

CDibLookApp::CDibLookApp()
{
  m_lpParams = NULL;
  m_autoread = FALSE;
}

CDibLookApp::~CDibLookApp()
{
  if (m_lpParams != NULL)
      delete m_lpParams;
}


/////////////////////////////////////////////////////////////////////////////
// The one and only CDibLookApp object

CDibLookApp NEAR theApp;

/////////////////////////////////////////////////////////////////////////////
// CDibLookApp initialization

BOOL CDibLookApp::InitInstance()
{
  // Standard initialization
  //  (if you are not using these features and wish to reduce the size
  // of your final executable, you should remove the following initialization
  SetDialogBkColor();        // set dialog background color
  LoadStdProfileSettings();  // Load standard INI file options (including MRU)

  // Register document templates which serve as connection between
  //  documents and views.  Views are contained in the specified view

  AddDocTemplate(new CMultiDocTemplate(IDR_DIBTYPE,
          RUNTIME_CLASS(CDibDoc),
          RUNTIME_CLASS(CMyChildWnd),        // I replace CMDIChildWnd
          RUNTIME_CLASS(CDibView)));

  // create main MDI Frame window
  CMainFrame* pMainFrame = new CMainFrame;
  if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
      return FALSE;
  pMainFrame->ShowWindow(m_nCmdShow);
  pMainFrame->UpdateWindow();
  m_pMainWnd = pMainFrame;

  // enable file manager drag/drop and DDE Execute open
  m_pMainWnd->DragAcceptFiles();
  EnableShellOpen();
  RegisterShellFileTypes();

    // As a test, save a global copy of command line args
    // global_cmd_line_args = m_lpCmdLine;
    m_lpParams = new SignerParams(m_lpCmdLine);

    // DEBUG: display the command line before we parse it.
    // AfxMessageBox(m_lpCmdLine);

    // simple command line parsing
    if (m_lpParams->GetInputFilename() == NULL)
    {
        // create a new (empty) document
        // OnFileNew();
    }
    else if ((m_lpCmdLine[0] == '-' || m_lpCmdLine[0] == '/') &&
        (m_lpCmdLine[1] == 'e' || m_lpCmdLine[1] == 'E'))
    {
        // program launched embedded - wait for DDE or OLE open
    }
    else
    {
        // open an existing document
        OpenDocumentFile(m_lpParams->GetInputFilename());
    }

    // Try adding another window.
    //pMainFrame->OnWindowNew();    fails: this is a protected member.
    // pMainFrame->SendMessage(ID_WINDOW_NEW);
    //pMainFrame->MyOnWindowNewTest();

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg() : CDialog(CAboutDlg::IDD)
        {
            //{{AFX_DATA_INIT(CAboutDlg)
            //}}AFX_DATA_INIT
        }

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CDibLookApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CDibLookApp commands
```

SIGNER.H

```
// signer.h : main header file for the SIGNER application
//
```

```
#ifndef __AFXWIN_H__
  #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

#include "params.h"

#define WM_DOREALIZE   (WM_USER + 0)

/////////////////////////////////////////////////////////////////////////////
// CDibLookApp:
// See diblook.cpp for the implementation of this class
//

class CDibLookApp : public CWinApp
{
public:
  CDibLookApp();
  ~CDibLookApp();

  // Create a command line parameter object.
  SignerParams *m_lpParams;
  SignerParams *getParams(void) {return m_lpParams;}

  BOOL m_autoread;


// Overrides
  virtual BOOL InitInstance();

// Implementation

  //{{AFX_MSG(CDibLookApp)
  afx_msg void OnAppAbout();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

SIGNER.RC

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

/////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

/////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.rc""\r\n"
    "#include ""afxprint.rc""\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

/////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "RES\\DIBLOOK.ICO"
IDR_DIBTYPE             ICON    DISCARDABLE     "RES\\DIBDOC.ICO"

/////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "RES\\TOOLBAR.BMP"

/////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",              ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",          ID_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "P&rint Setup...",           ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",               ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                     ID_APP_EXIT
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                  ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",               ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About SIGNER..",           ID_APP_ABOUT
    END
END

IDR_DIBTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",              ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",          ID_FILE_OPEN
        MENUITEM "&Close",                    ID_FILE_CLOSE
        MENUITEM "Save &As...",               ID_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\tCtrl+P",         ID_FILE_PRINT
        MENUITEM "Print Pre&view",            ID_FILE_PRINT_PREVIEW
        MENUITEM "P&rint Setup...",           ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",               ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                     ID_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",             ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",              ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",             ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",            ID_EDIT_PASTE
    END
    POPUP "&Actions"
    BEGIN
        MENUITEM "&Sign...",                  ID_SETTINGS_SIGNER
        MENUITEM "&Align...",                 ID_SETTINGS_ALIGN
        MENUITEM "&Read...",                  ID_SETTINGS_READER
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&New Window",               ID_WINDOW_NEW
        MENUITEM "&Cascade",                  ID_WINDOW_CASCADE
        MENUITEM "&Tile",                     ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",            ID_WINDOW_ARRANGE
    END
```

```
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
        MENUITEM SEPARATOR
        MENUITEM "Signed Image",                ID_VIEW_SIGNED
        MENUITEM "Unsigned Image",              ID_VIEW_UNSIGNED
        MENUITEM "Code Pattern",                ID_VIEW_SNOWY_IMAGE
        MENUITEM "Status",                      ID_VIEW_STATUS
    END
    POPUP "&Options"
    BEGIN
        MENUITEM "Auto-read After Signing",     ID_SETTINGS_AUTOREAD
        MENUITEM "Registry...",                 ID_SETTINGS_REGISTRY, GRAYED
        MENUITEM "Auto-print Report",           ID_SETTINGS_AUTOPRINT
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About SIGNER..",             ID_APP_ABOUT
    END
END

/////////////////////////////////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "N",            ID_FILE_NEW,            VIRTKEY, CONTROL
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL
    "P",            ID_FILE_PRINT,          VIRTKEY, CONTROL
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT
END

/////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  34, 22, 216, 91
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,18,20
    LTEXT           "Digimarc Win32 Signer Version 0.24",IDC_STATIC,40,10,
                    127,8
    LTEXT           "Copyright - 1995, 1996",IDC_STATIC,40,40,119,8
    DEFPUSHBUTTON   "OK",IDOK,176,6,32,14,WS_GROUP
    LTEXT           "For internal evaluation only.",IDC_STATIC,40,55,100,10
    LTEXT           "Rev 04/10/96",IDC_STATIC,40,25,57,8
END

IDD_PARAMS_DIALOG DIALOG DISCARDABLE  0, 0, 232, 179
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Signer Controls Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,45,144,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,135,144,50,14
    EDITTEXT        IDC_MESSAGE,6,17,221,15,ES_AUTOHSCROLL
    LTEXT           "Key:",IDC_STATIC,8,48,30,8
    EDITTEXT        IDC_EDIT_KEY,92,45,40,13,ES_AUTOHSCROLL
    LTEXT           "Gain:",IDC_STATIC,8,70,30,9
    EDITTEXT        IDC_EDIT_GAIN,92,67,40,13,ES_AUTOHSCROLL
    LTEXT           "Bump Size:",IDC_STATIC,8,93,44,8
    EDITTEXT        IDC_BUMP_SIZE,92,89,40,13,ES_AUTOHSCROLL
    LTEXT           "Message:",IDC_MESSAGE_LABEL,6,5,58,10
    LTEXT           "Detail Gain:",IDC_STATIC,8,115,60,8
    EDITTEXT        IDC_DETAIL_SCALE,92,111,40,14,ES_AUTOHSCROLL
END

IDD_READ_DIALOG DIALOG DISCARDABLE  0, 0, 152, 200
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Reader Controls Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,8,160,50,15
    PUSHBUTTON      "Cancel",IDCANCEL,80,160,50,14
    LTEXT           "Key:",IDC_STATIC,15,45,40,8
    EDITTEXT        IDC_READ_KEY,93,41,26,13,ES_AUTOHSCROLL
    LTEXT           "Message Length:",IDC_STATIC,15,65,72,8
    EDITTEXT        IDC_READ_LENGTH,93,62,26,13,ES_AUTOHSCROLL
    LTEXT           "Gain:",IDC_STATIC,15,85,55,8
    EDITTEXT        IDC_READ_GAIN,93,83,26,13,ES_AUTOHSCROLL
    LTEXT           "Bump Size:",IDC_STATIC,15,107,58,8
    EDITTEXT        IDC_BUMP_SIZE,93,104,26,14,ES_AUTOHSCROLL
    LTEXT           "Enter parameters to read a Digimarc message from active window.",
                    IDC_STATIC,6,8,131,25
    LTEXT           "Detail Gain:",IDC_STATIC,15,129,63,8
    EDITTEXT        IDC_DETAIL_LUT_SCALE,93,126,26,14,ES_AUTOHSCROLL
END

/////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "Digimarc Signer Application"
    IDR_DIBTYPE             "\n\nSIGNER Document\nBMP Files
(*.bmp)\n.bmp\nSignerFileType\nSIGNER File Type"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "Digimarc Signer Application"
    AFX_IDS_IDLEMESSAGE     "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW             "Create a new document"
    ID_FILE_OPEN            "Open an existing document"
    ID_FILE_CLOSE           "Close the active document"
    ID_FILE_SAVE            "Save the active document"
    ID_FILE_SAVE_AS         "Save the signed image with a new name"
    ID_FILE_PAGE_SETUP      "Change the printing options"
    ID_FILE_PRINT_SETUP     "Change the printer and printing options"
    ID_FILE_PRINT           "Print the active document"
    ID_FILE_PRINT_PREVIEW   "Display full pages"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT            "Display program information, version number and copyright"
    ID_APP_EXIT             "Quit the application; prompts to save documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE            "Switch to the next window pane"
    ID_PREV_PANE            "Switch back to the previous window pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW           "Open another window for the active document"
    ID_WINDOW_ARRANGE       "Arrange icons at the bottom of the window"
    ID_WINDOW_CASCADE       "Arrange windows so they overlap"
    ID_WINDOW_TILE_HORZ     "Arrange windows as non-overlapping tiles"
    ID_WINDOW_TILE_VERT     "Arrange windows as non-overlapping tiles"
    ID_WINDOW_SPLIT         "Split the active window into panes"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR           "Erase the selection"
```

```
        ID_EDIT_CLEAR_ALL       "Erase everything"
        ID_EDIT_COPY            "Copy the selection and puts it on the Clipboard"
        ID_EDIT_CUT             "Cut the selection and puts it on the Clipboard"
        ID_EDIT_FIND            "Find the specified text"
        ID_EDIT_PASTE           "Insert Clipboard contents"
        ID_EDIT_REPEAT          "Repeat the last action"
        ID_EDIT_REPLACE         "Replace specific text with different text"
        ID_EDIT_SELECT_ALL      "Select the entire document"
        ID_EDIT_UNDO            "Undo the last action"
        ID_EDIT_REDO            "Redo the previously undone action"
END

STRINGTABLE DISCARDABLE
BEGIN
        ID_VIEW_TOOLBAR         "Show or hide the toolbar"
        ID_VIEW_STATUS_BAR      "Show or hide the status bar"
END

STRINGTABLE DISCARDABLE
BEGIN
        AFX_IDS_SCSIZE          "Change the window size"
        AFX_IDS_SCMOVE          "Change the window position"
        AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
        AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
        AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
        AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
        AFX_IDS_SCCLOSE         "Close the active window and prompts to save the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
        AFX_IDS_SCRESTORE       "Restore the window to normal size"
        AFX_IDS_SCTASKLIST      "Activate Task List"
        AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
        ID_EDIT_SETTINGS        "Edit parameters which control signing of images"
        ID_VIEW_SIGNED          "Display the signed image in this window."
        ID_VIEW_UNSIGNED        "View the unsigned image in this window."
        ID_VIEW_SNOW            "View the ""snowy image"" in this window"
        ID_VIEW_SNOWY_IMAGE     "View the snowy image in this window."
        ID_VIEW_STATUS          "View Signer/Reader status information in this window."
        ID_SETTINGS_SIGNER      "Sign the original image"
        ID_SETTINGS_READER      "Read the Digimarc message from the active image window"
        ID_SETTINGS_REGISTRY    "Set the name of the registry file."
        ID_SETTINGS_AUTOPRINTREPORT
                                "When checked, report is printed when file is saved."
        ID_SETTINGS_AUTOPRINT   "Automatically print status report when file is saved."
        ID_OPTIONS_AUTOREAD     "Automatically read the image after signing"
        ID_SETTINGS_AUTOREAD    "Automatically read the image after signing."
        ID_CONTROLS_ALIGN       "Use the image alignment feature."
        ID_SETTINGS_ALIGN       "Align the original image in preparation for reading"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#include "afxres.rc"
#include "afxprint.rc"

/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

SIGNERWI.MAK

```
# Microsoft Developer Studio Generated NMAKE File, Format Version 4.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Application" 0x0101

!IF "$(CFG)" == ""
CFG=Signer - Win32 Debug
!MESSAGE No configuration specified.  Defaulting to Signer - Win32 Debug.
!ENDIF

!IF "$(CFG)" != "Signer - Win32 Release" && "$(CFG)" != "Signer - Win32 Debug"
!MESSAGE Invalid configuration "$(CFG)" specified.
!MESSAGE You can specify a configuration when running NMAKE on this makefile
!MESSAGE by defining the macro CFG on the command line.  For example:
!MESSAGE
!MESSAGE NMAKE /f "SignerWin32.mak" CFG="Signer - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Signer - Win32 Release" (based on "Win32 (x86) Application")
!MESSAGE "Signer - Win32 Debug" (based on "Win32 (x86) Application")
!MESSAGE
!ERROR An invalid configuration is specified.
!ENDIF

!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF
################################################################################
# Begin Project
# PROP Target_Last_Scanned "Signer - Win32 Debug"
MTL=mktyplib.exe
RSC=rc.exe
CPP=cl.exe

!IF  "$(CFG)" == "Signer - Win32 Release"

# PROP BASE Use_MFC 1
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 1
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
OUTDIR=.\Release
INTDIR=.\Release

ALL : "$(OUTDIR)\SignerWin32.exe" "$(OUTDIR)\SignerWin32.bsc"

CLEAN :
	-@erase ".\Release\SignerWin32.bsc"
	-@erase ".\Release\Mainfrm.sbr"
	-@erase ".\Release\Sign.sbr"
	-@erase ".\Release\Signdoc.sbr"
	-@erase ".\Release\Coxkey.sbr"
	-@erase ".\Release\Parmsdlg.sbr"
	-@erase ".\Release\Fft.sbr"
	-@erase ".\Release\Stdafx.sbr"
	-@erase ".\Release\Mychildw.sbr"
	-@erase ".\Release\Packmsg.sbr"
	-@erase ".\Release\Signview.sbr"
	-@erase ".\Release\Myfile.sbr"
	-@erase ".\Release\Image.sbr"
	-@erase ".\Release\Params.sbr"
	-@erase ".\Release\Signer.sbr"
	-@erase ".\Release\Align.sbr"
	-@erase ".\Release\Read.sbr"
	-@erase ".\Release\Dibapi.sbr"
	-@erase ".\Release\Readdlg.sbr"
	-@erase ".\Release\SignerWin32.exe"
	-@erase ".\Release\Params.obj"
	-@erase ".\Release\Signer.obj"
	-@erase ".\Release\Align.obj"
	-@erase ".\Release\Read.obj"
	-@erase ".\Release\Dibapi.obj"
	-@erase ".\Release\Readdlg.obj"
	-@erase ".\Release\Mainfrm.obj"
	-@erase ".\Release\Sign.obj"
	-@erase ".\Release\Signdoc.obj"
	-@erase ".\Release\Coxkey.obj"
	-@erase ".\Release\Parmsdlg.obj"
	-@erase ".\Release\Fft.obj"
	-@erase ".\Release\Stdafx.obj"
	-@erase ".\Release\Mychildw.obj"
	-@erase ".\Release\Packmsg.obj"
	-@erase ".\Release\Signview.obj"
	-@erase ".\Release\Myfile.obj"
	-@erase ".\Release\Image.obj"
	-@erase ".\Release\Signer.res"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

# ADD BASE CPP /nologo /MT /W3 /GX /O1 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS" /FR /YX
/c
# ADD CPP /nologo /MT /W3 /GX /O1 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS" /FR /YX /c
CPP_PROJ=/nologo /MT /W3 /GX /O1 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D\
 "_MBCS" /FR"$(INTDIR)/" /Fp"$(INTDIR)/SignerWin32.pch" /YX /Fo"$(INTDIR)/" /c
```

```
CPP_OBJS=.\Release/
CPP_SBRS=.\Release/
# ADD BASE MTL /nologo /D "NDEBUG" /win32
# ADD MTL /nologo /D "NDEBUG" /win32
MTL_PROJ=/nologo /D "NDEBUG" /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
RSC_PROJ=/l 0x409 /fo"$(INTDIR)/Signer.res" /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o"$(OUTDIR)/SignerWin32.bsc"
BSC32_SBRS= \
  "$(INTDIR)/Mainfrm.sbr" \
  "$(INTDIR)/Sign.sbr" \
  "$(INTDIR)/Signdoc.sbr" \
  "$(INTDIR)/Coxkey.sbr" \
  "$(INTDIR)/Parmsdlg.sbr" \
  "$(INTDIR)/Fft.sbr" \
  "$(INTDIR)/Stdafx.sbr" \
  "$(INTDIR)/Mychildw.sbr" \
  "$(INTDIR)/Packmsg.sbr" \
  "$(INTDIR)/Signview.sbr" \
  "$(INTDIR)/Myfile.sbr" \
  "$(INTDIR)/Image.sbr" \
  "$(INTDIR)/Params.sbr" \
  "$(INTDIR)/Signer.sbr" \
  "$(INTDIR)/Align.sbr" \
  "$(INTDIR)/Read.sbr" \
  "$(INTDIR)/Dibapi.sbr" \
  "$(INTDIR)/Readdlg.sbr"

"$(OUTDIR)\SignerWin32.bsc" : "$(OUTDIR)" $(BSC32_SBRS)
    $(BSC32) @<<
  $(BSC32_FLAGS) $(BSC32_SBRS)
<<

LINK32=link.exe
# ADD BASE LINK32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /machine:IX86
# ADD LINK32 oldnames.lib /nologo /stack:0x4800 /subsystem:windows /machine:IX86
# SUBTRACT LINK32 /profile /debug
LINK32_FLAGS=oldnames.lib /nologo /stack:0x4800 /subsystem:windows\
 /incremental:no /pdb:"$(OUTDIR)/SignerWin32.pdb" /machine:IX86\
 /def:".\Signer.def" /out:"$(OUTDIR)/SignerWin32.exe"
DEF_FILE= \
  ".\Signer.def"
LINK32_OBJS= \
  "$(INTDIR)/Params.obj" \
  "$(INTDIR)/Signer.obj" \
  "$(INTDIR)/Align.obj" \
  "$(INTDIR)/Read.obj" \
  "$(INTDIR)/Dibapi.obj" \
  "$(INTDIR)/Readdlg.obj" \
  "$(INTDIR)/Mainfrm.obj" \
  "$(INTDIR)/Sign.obj" \
  "$(INTDIR)/Signdoc.obj" \
  "$(INTDIR)/Coxkey.obj" \
  "$(INTDIR)/Parmsdlg.obj" \
  "$(INTDIR)/Fft.obj" \
  "$(INTDIR)/Stdafx.obj" \
  "$(INTDIR)/Mychildw.obj" \
  "$(INTDIR)/Packmsg.obj" \
  "$(INTDIR)/Signview.obj" \
  "$(INTDIR)/Myfile.obj" \
  "$(INTDIR)/Image.obj" \
  "$(INTDIR)/Signer.res"

"$(OUTDIR)\SignerWin32.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

# PROP BASE Use_MFC 1
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 1
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Target_Dir ""
OUTDIR=.\Debug
INTDIR=.\Debug

ALL : "$(OUTDIR)\SignerWin32.exe" "$(OUTDIR)\SignerWin32.bsc"

CLEAN :
    -@erase ".\Debug\vc40.pdb"
    -@erase ".\Debug\vc40.idb"
    -@erase ".\Debug\SignerWin32.bsc"
    -@erase ".\Debug\Dibapi.sbr"
    -@erase ".\Debug\Readdlg.sbr"
    -@erase ".\Debug\Myfile.sbr"
    -@erase ".\Debug\Mychildw.sbr"
    -@erase ".\Debug\Coxkey.sbr"
    -@erase ".\Debug\Signview.sbr"
    -@erase ".\Debug\Signer.sbr"
    -@erase ".\Debug\Stdafx.sbr"
    -@erase ".\Debug\Read.sbr"
    -@erase ".\Debug\Packmsg.sbr"
    -@erase ".\Debug\Fft.sbr"
    -@erase ".\Debug\Sign.sbr"
    -@erase ".\Debug\Image.sbr"
    -@erase ".\Debug\Parmsdlg.sbr"
    -@erase ".\Debug\Mainfrm.sbr"
    -@erase ".\Debug\Signdoc.sbr"
    -@erase ".\Debug\Align.sbr"
    -@erase ".\Debug\Params.sbr"
    -@erase ".\Debug\SignerWin32.exe"
    -@erase ".\Debug\Params.obj"
    -@erase ".\Debug\Dibapi.obj"
    -@erase ".\Debug\Readdlg.obj"
    -@erase ".\Debug\Myfile.obj"
    -@erase ".\Debug\Mychildw.obj"
    -@erase ".\Debug\Coxkey.obj"
    -@erase ".\Debug\Signview.obj"
    -@erase ".\Debug\Signer.obj"
    -@erase ".\Debug\Stdafx.obj"
    -@erase ".\Debug\Read.obj"
    -@erase ".\Debug\Packmsg.obj"
    -@erase ".\Debug\Fft.obj"
    -@erase ".\Debug\Sign.obj"
    -@erase ".\Debug\Image.obj"
    -@erase ".\Debug\Parmsdlg.obj"
    -@erase ".\Debug\Mainfrm.obj"
    -@erase ".\Debug\Signdoc.obj"
    -@erase ".\Debug\Align.obj"
    -@erase ".\Debug\Signer.res"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D
"_MBCS" /FR /YX /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_MBCS" /FR
/YX /c
CPP_PROJ=/nologo /MTd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"\
 /D "_MBCS" /FR"$(INTDIR)/" /Fp"$(INTDIR)/SignerWin32.pch" /YX /Fo"$(INTDIR)/"\
 /Fd"$(INTDIR)/" /c
CPP_OBJS=.\Debug/
CPP_SBRS=.\Debug/
# ADD BASE MTL /nologo /D "_DEBUG" /win32
# ADD MTL /nologo /D "_DEBUG" /win32
MTL_PROJ=/nologo /D "_DEBUG" /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG"
# ADD RSC /l 0x409 /d "_DEBUG"
RSC_PROJ=/l 0x409 /fo"$(INTDIR)/Signer.res" /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o"$(OUTDIR)/SignerWin32.bsc"
BSC32_SBRS= \
    "$(INTDIR)/Dibapi.sbr" \
    "$(INTDIR)/Readdlg.sbr" \
    "$(INTDIR)/Myfile.sbr" \
    "$(INTDIR)/Mychildw.sbr" \
    "$(INTDIR)/Coxkey.sbr" \
    "$(INTDIR)/Signview.sbr" \
    "$(INTDIR)/Signer.sbr" \
    "$(INTDIR)/Stdafx.sbr" \
    "$(INTDIR)/Read.sbr" \
    "$(INTDIR)/Packmsg.sbr" \
    "$(INTDIR)/Fft.sbr" \
    "$(INTDIR)/Sign.sbr" \
    "$(INTDIR)/Image.sbr" \
    "$(INTDIR)/Parmsdlg.sbr" \
    "$(INTDIR)/Mainfrm.sbr" \
    "$(INTDIR)/Signdoc.sbr" \
    "$(INTDIR)/Align.sbr" \
    "$(INTDIR)/Params.sbr"

"$(OUTDIR)\SignerWin32.bsc" : "$(OUTDIR)" $(BSC32_SBRS)
    $(BSC32) @<<
  $(BSC32_FLAGS) $(BSC32_SBRS)
<<

LINK32=link.exe
```

```
LINK32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /debug /machine:IX86
32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /profile /debug /machine:IX86
LAGS=oldnames.lib /nologo /stack:0x2800 /subsystem:windows /profile\
 /machine:IX86 /def:".\Signer.def" /out:"$(OUTDIR)/SignerWin32.exe"
ILE= \
\Signer.def"
K32_OBJS= \
 "$(INTDIR)/Params.obj" \
 "$(INTDIR)/Dibapi.obj" \
 "$(INTDIR)/Readdlg.obj" \
 "$(INTDIR)/Myfile.obj" \
 "$(INTDIR)/Mychildw.obj" \
 "$(INTDIR)/Coxkey.obj" \
 "$(INTDIR)/Signview.obj" \
 "$(INTDIR)/Signer.obj" \
 "$(INTDIR)/Stdafx.obj" \
 "$(INTDIR)/Read.obj" \
 "$(INTDIR)/Packmsg.obj" \
 "$(INTDIR)/Fft.obj" \
 "$(INTDIR)/Sign.obj" \
 "$(INTDIR)/Image.obj" \
 "$(INTDIR)/Parmsdlg.obj" \
 "$(INTDIR)/Mainfrm.obj" \
 "$(INTDIR)/Signdoc.obj" \
 "$(INTDIR)/Align.obj" \
 "$(INTDIR)/Signer.res"

"$(OUTDIR)\SignerWin32.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ENDIF

.c{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cpp{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cxx{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.c{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

.cpp{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

.cxx{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

################################################################################
# Begin Target

# Name "Signer - Win32 Release"
# Name "Signer - Win32 Debug"

!IF  "$(CFG)" == "Signer - Win32 Release"

!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

!ENDIF

################################################################################
# Begin Source File

SOURCE=.\Coxkey.cpp
DEP_CPP_COXKE=\
  ".\Coxkey.h"\
  ".\Dibapi.h"\
  ".\Stdafx.h"\


"$(INTDIR)\Coxkey.obj" : $(SOURCE) $(DEP_CPP_COXKE) "$(INTDIR)"

"$(INTDIR)\Coxkey.sbr" : $(SOURCE) $(DEP_CPP_COXKE) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Dibapi.cpp
DEP_CPP_DIBAP=\
  ".\Stdafx.h"\
  ".\Dibapi.h"\


"$(INTDIR)\Dibapi.obj" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR)"

"$(INTDIR)\Dibapi.sbr" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Image.cpp
DEP_CPP_IMAGE=\
    ".\Image.h"\
    ".\Dibapi.h"\
    ".\Stdafx.h"\


"$(INTDIR)\Image.obj" : $(SOURCE) $(DEP_CPP_IMAGE) "$(INTDIR)"

"$(INTDIR)\Image.sbr" : $(SOURCE) $(DEP_CPP_IMAGE) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Mainfrm.cpp

!IF  "$(CFG)" == "Signer - Win32 Release"

DEP_CPP_MAINF=\
    ".\Stdafx.h"\
    ".\Signer.h"\
    ".\Mainfrm.h"\
    ".\Params.h"\


"$(INTDIR)\Mainfrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"

"$(INTDIR)\Mainfrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"

!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

DEP_CPP_MAINF=\
    ".\Stdafx.h"\
    ".\Signer.h"\
    ".\Mainfrm.h"\


"$(INTDIR)\Mainfrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"

"$(INTDIR)\Mainfrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"

!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Myfile.cpp
DEP_CPP_MYFIL=\
    ".\Stdafx.h"\
    ".\Dibapi.h"\


"$(INTDIR)\Myfile.obj" : $(SOURCE) $(DEP_CPP_MYFIL) "$(INTDIR)"

"$(INTDIR)\Myfile.sbr" : $(SOURCE) $(DEP_CPP_MYFIL) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Packmsg.cpp
DEP_CPP_PACKM=\
    ".\Stdafx.h"\
    ".\packmsg.h"\


"$(INTDIR)\Packmsg.obj" : $(SOURCE) $(DEP_CPP_PACKM) "$(INTDIR)"

"$(INTDIR)\Packmsg.sbr" : $(SOURCE) $(DEP_CPP_PACKM) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Params.cpp
DEP_CPP_PARAM=\
```

```
  ".\Params.h"\
  ".\Stdafx.h"\


"$(INTDIR)\Params.obj" : $(SOURCE) $(DEP_CPP_PARAM) "$(INTDIR)"

"$(INTDIR)\Params.sbr" : $(SOURCE) $(DEP_CPP_PARAM) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Parmsdlg.cpp

!IF  "$(CFG)" == "Signer - Win32 Release"

DEP_CPP_PARMS=\
  ".\Stdafx.h"\
  ".\Signer.h"\
  ".\Parmsdlg.h"\
  ".\Params.h"\


"$(INTDIR)\Parmsdlg.obj" : $(SOURCE) $(DEP_CPP_PARMS) "$(INTDIR)"

"$(INTDIR)\Parmsdlg.sbr" : $(SOURCE) $(DEP_CPP_PARMS) "$(INTDIR)"


!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

DEP_CPP_PARMS=\
  ".\Stdafx.h"\
  ".\Signer.h"\
  ".\Parmsdlg.h"\


"$(INTDIR)\Parmsdlg.obj" : $(SOURCE) $(DEP_CPP_PARMS) "$(INTDIR)"

"$(INTDIR)\Parmsdlg.sbr" : $(SOURCE) $(DEP_CPP_PARMS) "$(INTDIR)"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Read.cpp
DEP_CPP_READ_=\
  ".\Read.h"\
  ".\Sign.h"\
  ".\fft.h"\
  ".\Stdafx.h"\


"$(INTDIR)\Read.obj" : $(SOURCE) $(DEP_CPP_READ_) "$(INTDIR)"

"$(INTDIR)\Read.sbr" : $(SOURCE) $(DEP_CPP_READ_) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Sign.cpp
DEP_CPP_SIGN_=\
  ".\Sign.h"\
  ".\Stdafx.h"\


"$(INTDIR)\Sign.obj" : $(SOURCE) $(DEP_CPP_SIGN_) "$(INTDIR)"

"$(INTDIR)\Sign.sbr" : $(SOURCE) $(DEP_CPP_SIGN_) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Stdafx.cpp
DEP_CPP_STDAF=\
  ".\Stdafx.h"\


"$(INTDIR)\Stdafx.obj" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"

"$(INTDIR)\Stdafx.sbr" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Signer.rc
DEP_RSC_SIGNE=\
    ".\RES\DIBLOOK.ICO"\
    ".\RES\DIBDOC.ICO"\
    ".\RES\TOOLBAR.BMP"\


"$(INTDIR)\Signer.res" : $(SOURCE) $(DEP_RSC_SIGNE) "$(INTDIR)"
   $(RSC) $(RSC_PROJ) $(SOURCE)


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Signer.cpp
DEP_CPP_SIGNER=\
    ".\Stdafx.h"\
    ".\Signer.h"\
    ".\Mainfrm.h"\
    ".\Signdoc.h"\
    ".\Signview.h"\
    ".\Mychildw.h"\
    ".\Params.h"\
    ".\Dibapi.h"\
    ".\packmsg.h"\
    ".\Image.h"\
    ".\Align.h"\


"$(INTDIR)\Signer.obj" : $(SOURCE) $(DEP_CPP_SIGNER) "$(INTDIR)"

"$(INTDIR)\Signer.sbr" : $(SOURCE) $(DEP_CPP_SIGNER) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Signdoc.cpp

!IF  "$(CFG)" == "Signer - Win32 Release"

DEP_CPP_SIGND=\
    ".\Stdafx.h"\
    ".\Signer.h"\
    ".\Signdoc.h"\
    ".\Signview.h"\
    ".\Coxkey.h"\
    ".\Image.h"\
    ".\Sign.h"\
    ".\Read.h"\
    ".\Align.h"\
    ".\Parmsdlg.h"\
    ".\readdlg.h"\
    ".\Mainfrm.h"\
    ".\Dibapi.h"\
    ".\packmsg.h"\
    ".\Params.h"\


"$(INTDIR)\Signdoc.obj" : $(SOURCE) $(DEP_CPP_SIGND) "$(INTDIR)"

"$(INTDIR)\Signdoc.sbr" : $(SOURCE) $(DEP_CPP_SIGND) "$(INTDIR)"


!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

DEP_CPP_SIGND=\
    ".\Stdafx.h"\
    ".\Signer.h"\
    ".\Signdoc.h"\
    ".\Signview.h"\
    ".\Coxkey.h"\
    ".\Image.h"\
    ".\Sign.h"\
    ".\Read.h"\
    ".\Align.h"\
    ".\Parmsdlg.h"\
    ".\readdlg.h"\
    ".\Mainfrm.h"\
    ".\Params.h"\
    ".\Dibapi.h"\
    ".\packmsg.h"\
```

```
"$(INTDIR)\Signdoc.obj" : $(SOURCE) $(DEP_CPP_SIGND) "$(INTDIR)"

"$(INTDIR)\Signdoc.sbr" : $(SOURCE) $(DEP_CPP_SIGND) "$(INTDIR)"

!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Signview.cpp
DEP_CPP_SIGNV=\
  ".\Stdafx.h"\
  ".\Signer.h"\
  ".\Signdoc.h"\
  ".\Signview.h"\
  ".\Dibapi.h"\
  ".\Mainfrm.h"\
  ".\Align.h"\
  ".\Params.h"\
  ".\packmsg.h"\
  ".\Image.h"\


"$(INTDIR)\Signview.obj" : $(SOURCE) $(DEP_CPP_SIGNV) "$(INTDIR)"

"$(INTDIR)\Signview.sbr" : $(SOURCE) $(DEP_CPP_SIGNV) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Mychildw.cpp

!IF  "$(CFG)" == "Signer - Win32 Release"

DEP_CPP_MYCHI=\
  ".\Stdafx.h"\
  ".\Signer.h"\
  ".\Mychildw.h"\
  ".\Params.h"\


"$(INTDIR)\Mychildw.obj" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"

"$(INTDIR)\Mychildw.sbr" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"


!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

DEP_CPP_MYCHI=\
  ".\Stdafx.h"\
  ".\Signer.h"\
  ".\Mychildw.h"\


"$(INTDIR)\Mychildw.obj" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"

"$(INTDIR)\Mychildw.sbr" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Readdlg.cpp

!IF  "$(CFG)" == "Signer - Win32 Release"

DEP_CPP_READD=\
  ".\Stdafx.h"\
  ".\Signer.h"\
  ".\readdlg.h"\
  ".\Params.h"\


"$(INTDIR)\Readdlg.obj" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"

"$(INTDIR)\Readdlg.sbr" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"


!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

DEP_CPP_READD=\
  ".\Stdafx.h"\
  ".\Signer.h"\
  ".\readdlg.h"\


"$(INTDIR)\Readdlg.obj" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"

"$(INTDIR)\Readdlg.sbr" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Signer.def

!IF  "$(CFG)" == "Signer - Win32 Release"

!ELSEIF  "$(CFG)" == "Signer - Win32 Debug"

!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Align.cpp

"$(INTDIR)\Align.obj" : $(SOURCE) "$(INTDIR)"

"$(INTDIR)\Align.sbr" : $(SOURCE) "$(INTDIR)"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Fft.cpp

"$(INTDIR)\Fft.obj" : $(SOURCE) "$(INTDIR)"

"$(INTDIR)\Fft.sbr" : $(SOURCE) "$(INTDIR)"


# End Source File
# End Target
# End Project
################################################################################
```

SIGNVIEW.CPP

```
//////////////////////////////////////////////////////////////////////////
// Signview.cpp
//
// Implementation of the CDibView class
//
//////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "signer.h"

#include "signdoc.h"
#include "signview.h"
#include "dibapi.h"
#include "mainfrm.h"
#include "Align.h"          // need to know about AlignStatus struct

#include <strstrea.h>
#include <iomanip.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

///////////////////////////////////////////////////////////////////////////
// CDibView
//
//
//////////////////////////////////////////////////////////////////////////
IMPLEMENT_DYNCREATE(CDibView, CScrollView)

BEGIN_MESSAGE_MAP(CDibView, CScrollView)
    //{{AFX_MSG_MAP(CDibView)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    ON_MESSAGE(WM_DOREALIZE, OnDoRealize)
```

```
  ON_COMMAND(ID_VIEW_SIGNED, OnViewSigned)
  ON_COMMAND(ID_VIEW_UNSIGNED, OnViewUnsigned)
  ON_COMMAND(ID_VIEW_SNOWY_IMAGE, OnViewSnowyImage)
  ON_COMMAND(ID_VIEW_STATUS, OnViewStatus)
  ON_UPDATE_COMMAND_UI(ID_VIEW_SIGNED, OnUpdateViewSigned)
  ON_UPDATE_COMMAND_UI(ID_VIEW_SNOWY_IMAGE, OnUpdateViewSnowyImage)
  ON_UPDATE_COMMAND_UI(ID_VIEW_STATUS, OnUpdateViewStatus)
  ON_UPDATE_COMMAND_UI(ID_VIEW_UNSIGNED, OnUpdateViewUnsigned)
  //}}AFX_MSG_MAP

  // Standard printing commands
  ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
  ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDibView()
//
// The constructor
//
//
/////////////////////////////////////////////////////////////////////////
CDibView::CDibView()
{
  m_viewType = ORIGINAL_VIEW;      // default type of view
  m_bThisViewActive = FALSE;       // View is initially inactive
  m_bDoResizeStatusView = FALSE;
}

/////////////////////////////////////////////////////////////////////////
// ~CDibView()
//
// The destructor.
//
/////////////////////////////////////////////////////////////////////////
CDibView::~CDibView()
{
}

/////////////////////////////////////////////////////////////////////////
// GetHDIB()
//
// Returns the HDIB (handle to the DIB) of the current view.  Note that
// it doesn't make sense to call this if the current view is the status
// view, or any other view which isn't displaying a DIB.
/////////////////////////////////////////////////////////////////////////
HDIB CDibView::GetHDIB(void)
{
  CDibDoc* pDoc = GetDocument();

  switch (m_viewType)
  {
      case ORIGINAL_VIEW:
          return pDoc->GetOriginalHDIB();
          break;
      case SIGNED_VIEW:
          return pDoc->GetSignedHDIB();
          break;
      case SNOWY_VIEW:
          return pDoc->GetSnowyHDIB();
          break;
      case REF_VIEW:
          return pDoc->GetRefHDIB();
          break;
      case ALIGNED_VIEW:
          return pDoc->GetAlignedHDIB();
///       case STATUS_VIEW:
///           return
      default:
          return pDoc->GetOriginalHDIB();
          break;
  }
}

/////////////////////////////////////////////////////////////////////////
// OnDraw()
//
// Given a pointer to a CDC (device context), this function is responsible
// for drawing the current view.
/////////////////////////////////////////////////////////////////////////
void CDibView::OnDraw(CDC* pDC)
{
  if (m_viewType == STATUS_VIEW)
  {
      DisplayStatus(pDC);
   }
   else
   {
      CDibDoc* pDoc = GetDocument();
      HDIB hDIB = GetHDIB();

      if (hDIB != NULL)
      {
          LPSTR lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);
          int cxDIB = (int) ::DIBWidth(lpDIB);          // Size of DIB - x
          int cyDIB = (int) ::DIBHeight(lpDIB);         // Size of DIB - y
          ::GlobalUnlock((HGLOBAL) hDIB);
          CRect rcDIB;
          rcDIB.top = rcDIB.left = 0;
          rcDIB.right = cxDIB;
          rcDIB.bottom = cyDIB;
          CRect rcDest;
          if (pDC->IsPrinting())    // printer DC
          {
              // get size of printer page (in pixels)
              int cxPage = pDC->GetDeviceCaps(HORZRES);
              int cyPage = pDC->GetDeviceCaps(VERTRES);
              // get printer pixels per inch
              int cxInch = pDC->GetDeviceCaps(LOGPIXELSX);
              int cyInch = pDC->GetDeviceCaps(LOGPIXELSY);

              //
              // Best Fit case -- create a rectangle which preserves
              // the DIB's aspect ratio, and fills the page horizontally.
              //
              // The formula in the "->bottom" field below calculates the Y
              // position of the printed bitmap, based on the size of the
              // bitmap, the width of the page, and the relative size of
              // a printed pixel (cyInch / cxInch).
              //
              rcDest.top = rcDest.left = 0;
              rcDest.bottom = (int)(((double)cyDIB * cxPage * cyInch)
                      / ((double)cxDIB * cxInch));
              rcDest.right = cxPage;
          }
          else   // not printer DC
          {
              rcDest = rcDIB;
          }
          ::PaintDIB(pDC->m_hDC, &rcDest, GetHDIB(), //pDoc->GetHDIB(),
              &rcDIB, pDoc->GetDocPalette());
      }
   }
}

/////////////////////////////////////////////////////////////////////////
// OnPreparePrinting()
//
/////////////////////////////////////////////////////////////////////////
BOOL CDibView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

/////////////////////////////////////////////////////////////////////////////
// CDibView commands

/////////////////////////////////////////////////////////////////////////
// OnDoRealize()
//
/////////////////////////////////////////////////////////////////////////
LRESULT CDibView::OnDoRealize(WPARAM wParam, LPARAM)
{
    ASSERT(wParam != NULL);
    CDibDoc* pDoc = GetDocument();
    //if (pDoc->GetHDIB() == NULL)
    if (GetHDIB() == NULL)
        return 0L;  // must be a new document

    CPalette* pPal = pDoc->GetDocPalette();
    if (pPal != NULL)
    {
        CMainFrame* pAppFrame = (CMainFrame*) AfxGetApp()->m_pMainWnd;
        ASSERT(pAppFrame->IsKindOf(RUNTIME_CLASS( CMainFrame )));

        CClientDC appDC(pAppFrame);
        // All views but one should be a background palette.
        // wParam contains a handle to the active view, so the SelectPalette
        // bForceBackground flag is FALSE only if wParam == m_hWnd (this view)
        CPalette* oldPalette = appDC.SelectPalette(pPal, ((HWND)wParam) != m_hWnd);

        if (oldPalette != NULL)
        {
          UINT nColorsChanged = appDC.RealizePalette();
          if (nColorsChanged > 0)
              pDoc->UpdateAllViews(NULL);
          appDC.SelectPalette(oldPalette, TRUE);
        }
        else
```

```
    {
        TRACE0("\tSelectPalette failed in CDibView::OnPaletteChanged\n");
    }
  }

  return 0L;
}

/////////////////////////////////////////////////////////////////////////////
// OnInitialUpdate()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnInitialUpdate()
{
  CScrollView::OnInitialUpdate();
  ASSERT(GetDocument() != NULL);

  SetScrollSizes(MM_TEXT, GetDocument()->GetDocSize());

  // Resize this view's window based on the size of the image.
  ResizeParentToFit();

  GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Original");
}

/////////////////////////////////////////////////////////////////////////////
// OnActivateView()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnActivateView(BOOL bActivate, CView* pActivateView,
                CView* pDeactiveView)
{
  CScrollView::OnActivateView(bActivate, pActivateView, pDeactiveView);

  if (bActivate)
  {
      m_bThisViewActive = TRUE;
      ASSERT(pActivateView == this);
      OnDoRealize((WPARAM)m_hWnd, 0);   // same as SendMessage(WM_DOREALIZE);
  }
  else
      m_bThisViewActive = FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// OnEditCopy()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnEditCopy()
{
  CDibDoc* pDoc = GetDocument();
  // Clean clipboard of contents, and copy the DIB.

  if (OpenClipboard())
  {
      BeginWaitCursor();
      EmptyClipboard();
      SetClipboardData (CF_DIB, CopyHandle((HANDLE) GetHDIB()) );  //pDoc->GetHDIB()) );
      CloseClipboard();
      EndWaitCursor();
  }
}


/////////////////////////////////////////////////////////////////////////////
// OnUpdateEditCopy()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
  pCmdUI->Enable(GetHDIB() != NULL);
}

/////////////////////////////////////////////////////////////////////////////
// OnEditPaste()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnEditPaste()
{
  HDIB hNewDIB = NULL;

  if (OpenClipboard())
  {
      BeginWaitCursor();

      hNewDIB = (HDIB) CopyHandle(::GetClipboardData(CF_DIB));

      CloseClipboard();

      if (hNewDIB != NULL)
      {
          CDibDoc* pDoc = GetDocument();
          pDoc->ReplaceHDIB(hNewDIB); // and free the old DIB
          pDoc->InitDIBData();     // set up new size & palette
          pDoc->SetModifiedFlag(TRUE);

          SetScrollSizes(MM_TEXT, pDoc->GetDocSize());
          OnDoRealize((WPARAM)m_hWnd,0);  // realize the new palette
          pDoc->UpdateAllViews(NULL);
      }
      EndWaitCursor();
  }
}

/////////////////////////////////////////////////////////////////////////////
// OnUpdateEditPaste()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(::IsClipboardFormatAvailable(CF_DIB));
}

/////////////////////////////////////////////////////////////////////////////
// OnViewSigned()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnViewSigned()
{
    CDibDoc* pDoc = GetDocument();

    m_viewType = SIGNED_VIEW;
    //pDoc->SetModifiedFlag(TRUE);

    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Signed");

    pDoc->UpdateAllViews(NULL);

}

/////////////////////////////////////////////////////////////////////////////
// OnViewUnsigned()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnViewUnsigned()
{
    CDibDoc* pDoc = GetDocument();

    m_viewType = ORIGINAL_VIEW;

    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Original");

    pDoc->UpdateAllViews(NULL);
}

/////////////////////////////////////////////////////////////////////////////
// OnViewSnowyImage()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnViewSnowyImage()
{
    CDibDoc* pDoc = GetDocument();

    m_viewType = SNOWY_VIEW;

    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Code Pattern");

    pDoc->UpdateAllViews(NULL);
}


/////////////////////////////////////////////////////////////////////////////
// OnViewStatus()
//
/////////////////////////////////////////////////////////////////////////////
void CDibView::OnViewStatus()
{
    CDibDoc* pDoc = GetDocument();

    m_viewType = STATUS_VIEW;

    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Status");

    pDoc->UpdateAllViews(NULL);

}
```

```
////////////////////////////////////////////////////////////////////////////
// SetViewType()
//
////////////////////////////////////////////////////////////////////////////
void CDibView::SetViewType(int type)
{
  CDibDoc* pDoc = GetDocument();

  switch (type)
  {
      case SIGNED_VIEW:
          m_viewType = SIGNED_VIEW;
          // Set the window title.
          GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Signed");
          break;

      case REF_VIEW:
          m_viewType = REF_VIEW;
          // Set the window title.
          GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Reference");
          break;

      case ALIGNED_VIEW:
          m_viewType = ALIGNED_VIEW;
          // Set the window title.
          GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Aligned");
          break;

      case STATUS_VIEW:
          m_viewType = STATUS_VIEW;

          // Set the window title.
          GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Status");
          break;

      default:
          // This is an error.
          // afxmessage
          break;
  }
}

////////////////////////////////////////////////////////////////////////////
// DisplayStatus()
//
////////////////////////////////////////////////////////////////////////////
void CDibView::DisplayStatus(CDC *pDC)
{
  CDibDoc* pDoc = GetDocument();
  TEXTMETRIC tm;
  CString text;
  CRect rect;
  CTime t;

  pDC->GetTextMetrics(&tm);

  int col = 20*tm.tmAveCharWidth;          // Offset to column where will write results
  int line = tm.tmHeight;
  ostrstream strm;

  createStatusStream(strm);

  int height;
  rect.top = 10;
  rect.left = 10;
  rect.right = 50 * tm.tmAveCharWidth;

  height = pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS | DT_CALCRECT);
  rect.bottom = height + 10;
  pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS);

  // Resize the scrollbars to fit the information it contains.

  CSize size = CSize(rect.right+10, rect.bottom);
  SetScrollSizes(MM_TEXT, size);

  if (m_bDoResizeStatusView)
  {
      m_bDoResizeStatusView = FALSE;
      ResizeStatusView(size);
  }

  // Once we call .str(), we must delete the allocated space.
  delete strm.str();

  return;
}
```

```
////////////////////////////////////////////////////////////////////////////
// createStatusStream()
//
// Insert a stream of characters in to the ostrstream passed in by
// the caller, which describes the status.  The state argument
// indicates our current program state, which influences what
// information is included in the stream data.
////////////////////////////////////////////////////////////////////////////
void CDibView::createStatusStream(ostrstream &strm)
{
    CDibDoc* pDoc = GetDocument();
    CTime t;
    int state = pDoc->GetState();
    PackedMsg *pMsg = pDoc->GetPackedMsg();

    strm << "\t\tSTATUS INFORMATION\n\n";

    switch (state)
    {
    case NO_IMAGE:            // This case shouldn't come up - no menu access.
        strm << "No image has been loaded.";
        break;

    case IMAGE_LOADED:
        strm << "\tThe loaded image hasn't been signed or read.";
        break;

    case IMAGE_SIGNED:
    case IMAGE_SIGNED_AND_VERIFIED:
    case IMAGE_SIGNED_AND_SAVED:
        strm << "Signer Status\n\n";
        strm << "\tOriginal Text:\t\t" << pMsg->getAsciiMsg() << "\n\n";

        strm << "\tMessage Length:\t\t" << pMsg->GetMsgLength() << "\n\n";

        strm << "\tGain Setting:\t\t" << pDoc->GetSignerParams()->GetGain() << "\n\n";

        // strm << "\tGamma:\t\t\t" << pDoc->GetSignerParams()->GetGamma() << "\n\n";

        strm << "\tKey:\t\t\t" << pDoc->GetSignerParams()->GetKey() << "\n\n";

        strm << "\tBump Size:\t\t" << pDoc->GetSignerParams()->GetBumpSize() << "\n\n";

        strm << "\tDetail Gain:\t\t" << pDoc->GetSignerParams()->GetLutScale() << "\n\n";

        strm << "\tChecksum:\t\t" << (unsigned) pMsg->GetSignerChecksum() << "\n\n";


        strm.fill('0');          // Change fill character for timestamps
        t = pDoc->GetSignerParams()->GetTimestamp();
        strm << "\tTime of Signing:\t\t";

        // Disable the 4270 warning.  This is a bug in MicroSoft's iomanip.h.
        // without this, the setw() io manipulator causes a warning.
#pragma warning(disable:4270)
        strm << setw(2) << t.GetHour() << ':'
             << setw(2) << t.GetMinute() << ':'
             << setw(2) << t.GetSecond() << "  ";
        strm << setw(2) << t.GetMonth() << '/'
             << setw(2) << t.GetDay() << '/'
             << setw(2) << (t.GetYear() - 1900);
        strm << "\n\n";
        strm.fill(' ');          // Reset fill character to default.

        // Put the warning level back to the default.
#pragma warning(default:4270)


        if (state == IMAGE_SIGNED_AND_SAVED)
            strm << "\tSigned image saved as:\t" << pDoc->GetFilename() << "\n\n";


        if (state == IMAGE_SIGNED_AND_VERIFIED)
        {
            strm << "Reader Status\n\n";

            strm << "\tRecognized Text:\t\t" << pMsg->getRecoveredAsciiMsg() << "\n\n";

            // Remove references to "super reader" for now
            //if (pDoc->GetSignerParams()->GetSuperReaderFlag())
            //  strm << "\tAlternative Reader:\t\t" << "On" << "\n\n";
            //else
            //  strm << "\tAlternative Reader:\t\t" << "Off" << "\n\n";

            // Adjust the floating point precision of the stream.
            strm.setf(ios::fixed, ios::floatfield);
            strm.precision(2);

            strm << "\tBit Success Rate (%):\t" << pMsg->GetPercentCorrect() << "\n\n";
```

```
            // Print crude metric.
            strm.precision(4);
            strm << "\tBit Estimator Std. Dev.:\t" << pDoc->GetMetric() << "\n\n";

            // Print range.
            strm << "\tBit Estimator Range:\t" << pDoc->GetRange() << "\n\n";

            strm << "\tEmbedded Checksum Read:\t" << (unsigned) pMsg->GetReaderChecksum()
                 << "\n\n";

            strm << "\tChecksum Calculated:\t" << (unsigned) pMsg->GetComputedReaderChecksum()
                 << "\n\n";
        }

        break;

    case SUSPECT_ALIGNED:
        AlignStatus     a_stats = pDoc->GetAlignStatus();   // Get the align status

        strm << "Aligned Image Status\n\n";

          // Adjust the floating point precision of the stream.
          strm.setf(ios::fixed, ios::floatfield);
          strm.precision(2);

        strm << "\tRotation applied to suspect:\t" << a_stats.rotation << "\n\n";
        strm << "\tTranslation (X, Y): \t(" << a_stats.x_trans
             << ',' << a_stats.y_trans << ")\n\n";
        strm << "\tScaling (X, Y):\t(" << a_stats.x_scale
             << ',' << a_stats.y_scale << ")\n\n";
        strm << "\tRefinement:\t\t" << a_stats.refinement << "\n\n";

        break;

    case SUSPECT_READ:
          strm << "Reader Status\n\n";

        strm << "\tAssumed Message Length:\t" << pMsg->GetMsgLength() << "\n\n";

          strm << "\tRecognized Text:\t\t" << pMsg->getRecoveredAsciiMsg() << "\n\n";

        strm << "\tAssumed Key:\t\t" << pDoc->GetSignerParams()->GetKey() << "\n\n";

        strm << "\tBump Size:\t\t" << pDoc->GetSignerParams()->GetBumpSize() << "\n\n";

        strm << "\tDetail Gain:\t\t" << pDoc->GetSignerParams()->GetLutScale() << "\n\n";

        // Remove references to "super reader" for now
        //if (pDoc->GetSignerParams()->GetSuperReaderFlag())
        //  strm << "\tAlternative Reader:\t\t" << "On" << "\n\n";
        //else
        //  strm << "\tAlternative Reader:\t\t" << "Off" << "\n\n";

          // Adjust the floating point precision of the stream.
          strm.setf(ios::fixed, ios::floatfield);
          strm.precision(2);

          // Print crude metric.
          strm.precision(4);
        strm << "\tBit Estimator Std. Dev.:\t" << pDoc->GetMetric() << "\n\n";

        // Print range.
        strm << "\tBit Estimator Range:\t" << pDoc->GetRange() << "\n\n";

        strm << "\tEmbedded Checksum Read:\t" << (unsigned) pMsg->GetReaderChecksum()
             << "\n\n";

        strm << "\tChecksum Calculated:\t" << (unsigned) pMsg->GetComputedReaderChecksum()
             << "\n\n";

        break;

      default:
        break;
  }

    // Add a null terminator (DrawText needs it).
  strm << '\0';
}
//////////////////////////////////////////////////////////////////////////////
// ResizeStatusView()
//
// Resizes the status view frame window.  The goal is to not
// move the upper left corner, and to not exceed the bounds of
// the MDI main frame window on the right or left borders.
//////////////////////////////////////////////////////////////////////////////
void CDibView::ResizeStatusView(CSize status_size)
{
  const int bar_height = 27;      // An emperically derived kludge

    CRect main_frame_rect, view_win_rect, view_client_rect;

    // Get the size of the *frame* window's client area
    AfxGetApp()->m_pMainWnd->GetWindowRect(&main_frame_rect);

    // Get current location and dimensions of the view window frame
    GetParentFrame()->GetWindowRect(&view_win_rect);

    GetClientRect(&view_client_rect);
    CSize view_client_size = CSize(view_client_rect.right,
                                   view_client_rect.bottom);

    // Expand view rect in x or y, if needed, to hold status size.
    int oversize;
    if ((oversize = status_size.cx - view_client_size.cx) > 0)
        view_win_rect.right += oversize;
    if ((oversize = status_size.cy - view_client_size.cy) > 0)
        view_win_rect.bottom += oversize;

    // But don't let the view window exceed the right or bottom of mainframe.
    if (view_win_rect.right > main_frame_rect.right)
        view_win_rect.right = main_frame_rect.right;
    if (view_win_rect.bottom > (main_frame_rect.bottom - bar_height))
        view_win_rect.bottom = main_frame_rect.bottom - bar_height;


    // Pure kludge here: without it window is moved down by the
    // height of the title bar -- I don't know why.
    CPoint y_shift = CPoint(0, bar_height);
    view_win_rect -= y_shift;

    // Convert from screen to coordinates of main frame client area.
    AfxGetApp()->m_pMainWnd->ScreenToClient(&view_win_rect);
    GetParentFrame()->MoveWindow(view_win_rect);

    ResizeParentToFit();
}

//////////////////////////////////////////////////////////////////////////////
// OnUpdateViewSigned()
//
//////////////////////////////////////////////////////////////////////////////
void CDibView::OnUpdateViewSigned(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if (m_viewType == SIGNED_VIEW)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

//////////////////////////////////////////////////////////////////////////////
// OnUpdateViewSnowyImage()
//
//////////////////////////////////////////////////////////////////////////////
void CDibView::OnUpdateViewSnowyImage(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if (m_viewType == SNOWY_VIEW)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

//////////////////////////////////////////////////////////////////////////////
// OnUpdateViewStatus()
//
//////////////////////////////////////////////////////////////////////////////
void CDibView::OnUpdateViewStatus(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if (m_viewType == STATUS_VIEW)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

//////////////////////////////////////////////////////////////////////////////
// OnUpdateViewUnsigned()
//
//////////////////////////////////////////////////////////////////////////////
void CDibView::OnUpdateViewUnsigned(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if (m_viewType == ORIGINAL_VIEW)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}
```

```
// signview.h : interface of the CDibView class
//

#include <strstrea.h>

// Here I define the differenct types of views.
#define UNKNOWN_VIEW  -1
#define SIGNED_VIEW       1
#define ORIGINAL_VIEW 2
#define SNOWY_VIEW        3
#define STATUS_VIEW       4
#define REF_VIEW       5       // reference image for alignment
#define ALIGNED_VIEW  6        // image after alignment completed


class CDibView : public CScrollView
{
public:
  CDibView();
  DECLARE_DYNCREATE(CDibView)

// Attributes
public:
  CDibDoc* GetDocument()
      {
          ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CDibDoc)));
          return (CDibDoc*) m_pDocument;
      }

private:
  int m_viewType;
  BOOL m_bThisViewActive;
  BOOL m_bDoResizeStatusView;

// Operations
public:

// Implementation
public:
  virtual ~CDibView();
  virtual void OnDraw(CDC* pDC);  // overridden to draw this view

  virtual void OnInitialUpdate();
  virtual void OnActivateView(BOOL bActivate, CView* pActivateView,
                CView* pDeactiveView);
  void SetViewType(int type);
  int GetViewType(void) {return m_viewType;}
  BOOL IsViewActive(void) {return m_bThisViewActive;}

  void DoResize(void) {m_bDoResizeStatusView = TRUE;}
    void ResizeStatusView(CSize status_size);

  // I need OnFilePrint to be accessible from outside.
  void OnFilePrint(void) {CScrollView::OnFilePrint();}

  void createStatusStream(ostrstream &strm);

  // Printing support
protected:
  virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);

private:
  HDIB GetHDIB(void);
  void CDibView::DisplayStatus(CDC *pDC);

// Generated message map functions
protected:
  //{{AFX_MSG(CDibView)
  afx_msg void OnEditCopy();
  afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
  afx_msg void OnEditPaste();
  afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
  afx_msg LRESULT OnDoRealize(WPARAM wParam, LPARAM lParam);  // user message
  afx_msg void OnViewSigned();
  afx_msg void OnViewUnsigned();
  afx_msg void OnViewSnowyImage();
  afx_msg void OnViewStatus();
  afx_msg void OnUpdateViewSigned(CCmdUI* pCmdUI);
  afx_msg void OnUpdateViewSnowyImage(CCmdUI* pCmdUI);
  afx_msg void OnUpdateViewStatus(CCmdUI* pCmdUI);
  afx_msg void OnUpdateViewUnsigned(CCmdUI* pCmdUI);
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

///////////////////////////////////////////////////////////////////////////
```



```
//////////////////////
// My experimental member function which
// builds a snowy image in place.
//
//////////////////////

void CDibDoc::MakeSnow(void)
{
    int     cxDIB, cyDIB;
    long    num_pixels, num_colors;
    LPSTR   lpDIB, lpSnowyDIB;            // Pointer to BITMAPINFOHEADER
    LPBITMAPINFOHEADER  lpDIBHdr, lpSnowyDIBHdr;
    LPSTR   lpDIBBits;      // Pointer to DIB bits
    char __huge *src_data, *dest_data;      // Huge ptrs for copying the image.


    HDIB hUnsignedDIB = GetHDIB();
    if (hUnsignedDIB == NULL)
        return;

    // Create space for the unsigned DIB for the snowy image.
    m_hSnowyDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, m_dwTotalDIBSize);
    if (m_hSnowyDIB == 0)
        return;



    // Here I follow the similar code in PaintDIB() of dibapi.cpp
    lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hUnsignedDIB);
    lpSnowyDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hSnowyDIB);

    src_data = (char __huge *) lpDIB;
    dest_data = (char __huge *) lpSnowyDIB;

    // Copy the BITMAPINFOHEADER, palette, and actual image byte data.
    for (image_byte = 0; image_byte < m_dwTotalDIBSize; image_byte++)
    {
        dest_data++ = src_data++;
    }



    lpDIBHdr = (LPBITMAPINFOHEADER) lpDIB;  // Ptr to bitmap info hdr at start of dib.

    // Get ptr to the snowy dib header space, and copy header into it.
    lpSnowyDIBHdr = (LPBITMAPINFOHEADER) lpSnowyDIB;
    *lpSnowyDIBHdr = *lpDIBHdr;


    lpDIBBits = ::FindDIBBits(lpDIB);
    lpSnowyDIBBits = ::FindDIBBits(lpSnowyDIB);

    src_data = (char __huge *) lpDIBBits;
    dest_data = (char __huge *) lpSnowyDIBBits;

    // Copy the actual image byte data.
    for (image_byte = 0; image_byte < m_dwTotalDIBSize; image_byte++)
    {
        dest_data++ = src_data++;
    }


    cxDIB = (int) ::DIBWidth(lpDIB);     // X size of DIB
    cyDIB = (int) ::DIBHeight(lpDIB);    // Y size of DIB

    num_pixels = (long) cxDIB *  cyDIB;

    num_colors = ::DIBNumColors(lpDIB);

    if (lpDIBHdr->biCompression != 0)
    {
        TRACE("Can't cope with compressed image (compression = %d)\n",
lpDIBHdr->biCompression);
        ::GlobalUnlock((HGLOBAL) hUnsignedDIB);
        return;
    }


    TRACE("width = %d, height = %d, num_pixels = %ld\n", cxDIB, cyDIB, num_pixels);
    TRACE("num_colors = %d\n", num_colors);

    if (num_colors == 0 || num_colors == 16)
    {
```

```
    TRACE("At this time, only build snowy image for 8 bit images\n");
    ::GlobalUnlock((HGLOBAL) hUnsignedDIB);
    return;
  }

  if (num_colors == 256)
  {
    CoXKey coXKey(1, (BITMAPINFO *) lpDIBHdr, lpDIBBits);
  }

  ::GlobalUnlock((HGLOBAL) hUnsignedDIB);
}
```

STDAFX.CPP

```
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.


// stdafx.cpp : source file that includes just the standard includes
//  stdafx.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

STDAFX.H

```
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#include <afxwin.h>         // MFC core and standard components
```

SIGN PUBLIC.CPP

```
//////////////////////////////////////////////////////////////////////////
// FILE: Sign_public.cpp
//
// DESCRIPTION:
// Core signing functions of the public digimarc technology.
// Started late April 1996
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//////////////////////////////////////////////////////////////////////////

#include "sign.h"
#include <math.h>
#include "stdafx.h"

#define SIGNATURE_BLOCK_DIMENSION 128
#define HIGHEST_GREY_VALUE 255
#define GRID_MINIMUM_GAIN -0.5
#define RED_DOG 0.33
#define GREEN_DOG 0.34
#define BLUE_DOG 0.33


// this function simply loads the floating point values of the bumps for a given "bump raster line"
// the output of this function (the bump array) should be roughly similar no matter
// what the bump size is or whether you're dealing with color or B&W
// REMEMBER: this function pads the ends on each side with one extra bump
int load_bump_array(
  float *bump,  // floating point bump array to be filled (output)
  unsigned char *data,  // input pixel data
    long xdim,  // number of bumps in this row (not pixels), add 2 for output
    long zdim,  // number of channels
    long bump_size,  // pixels per bump
    long jump_x,  // number of raw pixels between (xdim*bump_size) and entire image array x
dimension
    long overfill  // this tells the innards that the incoming bump array needs a copied value
into the first and last place
){
    unsigned char *pdata;
    long i,j,k;
    float *pbump,bump_squared = (float)bump_size * (float)bump_size;

    pdata = data;
    if(overfill)pbump = bump+1;
    else pbump = bump;
    if(zdim == 1){  // single channel
        if(bump_size == 1){
            for(j=0;j<xdim;j++)*(pbump++) = (float) *(pdata++);
        }
        else if(bump_size == 2){
            // zero out bump array
            memset(bump,0,(xdim+2)*sizeof(float));
            for(i=0;i<2;i++){
                if(overfill)pbump = bump+1;
                else pbump = bump;
                for(j=0;j<xdim;j++){
                    *pbump += (float) *(pdata++);
                    *(pbump++) += (float) *(pdata++);
                }
                pdata += jump_x;
            }
            if(overfill)pbump = bump+1;
            else pbump = bump;
            for(i=0;i<xdim;i++)*(pbump++) /= bump_squared;
        }
        else {
            // zero out bump array
            memset(bump,0,(xdim+2)*sizeof(float));
            for(i=0;i<bump_size;i++){
                if(overfill)pbump = bump+1;
                else pbump = bump;
                for(j=0;j<xdim;j++){
                    for(k=0;k<bump_size;k++)*pbump+=*(pdata++);
                    pbump++;
                }
                pdata += jump_x;
            }
            if(overfill)pbump = bump+1;
            else pbump = bump;
            for(i=0;i<xdim;i++)*(pbump++) /= bump_squared;
        }
    }
    else { // multi-channel, assume ONLY RGB and three channels at present
        float red = (float)RED_DOG,green=(float)GREEN_DOG,blue=(float)BLUE_DOG;
        if(bump_size == 1){ // this case is split off only for a X% speed Increase in
execution
            for(j=0;j<xdim;j++){
                *pbump = red * (float) *(pdata++);  // gimme an R
                *pbump += green * (float) *(pdata++);  // gimme a G
                *(pbump++) += blue * (float) *(pdata++);  // gimme a B
            }
        }
        else {
            // zero out bump array
            memset(bump,0,(xdim+2)*zdim*sizeof(float));
            for(i=0;i<bump_size;i++){
                if(overfill)pbump = bump+1;
                else pbump = bump;
                for(j=0;j<xdim;j++){
                    for(k=0;k<bump_size;k++){
                        *pbump += red * (float) *(pdata++);  // gimme an R
                        *pbump += green * (float) *(pdata++);  // gimme a G
                        *pbump += blue * (float) *(pdata++);  // gimme a B
                    }
                    pbump++;
                }
                pdata += zdim * jump_x;
            }
            if(overfill)pbump = bump+1;
            else pbump = bump;
            for(i=0;i<xdim;i++)*(pbump++) /= bump_squared;
        }
    }

    // fill the end two values
    if(overfill){
        bump[0]=bump[1];
        bump[xdim+1]=bump[xdim];
    }
```

```
  return(1);
}

////////////////////////////////////////////////////////////////////////////
// load_funky_lut()
//
// This function loads the scaling factor based on minimum linear funkiness
////////////////////////////////////////////////////////////////////////////
int load_funky_lut( float *funky_lut)   // explicitly written for 8 bit
{
  int i,status=1,detail_start,detail_stop;
  float length;

  float scale = (float)1.0;
  detail_start = 1;
  detail_stop = 50;
  length = (float)detail_stop - (float)detail_start;

  for(i=0;i<detail_start;i++)funky_lut[i]=(float)0.0;
  for(i=detail_start; i<detail_stop; i++)
  {
      funky_lut[i] =  scale*((float)(i-detail_start)/length);
  }
  for(i=detail_stop;i<512;i++)funky_lut[i]=funky_lut[detail_stop-1];

  return(status);
}


// this function associates a given row and column value of a bump in the
// standard signature block with A) the bit plane of the message associated with the bump,
// output in the 'message_bit_lut' variable array, and B) whether the '1' direction is up
// XOR_lut=1, or down, XOR_lut=0
// IMPORTANT: this also takes care of the basic XOR'ing operation between the message and
// the underlying code pattern (invert, don't invert)
int load_standard_message_block_lut(
  unsigned char *message,   // if this is NULL, return the un XOR'ed array (for reading)
  long message_length,
  unsigned char *control_message,  // this is the separate "always gotta be there" message
  long control_message_length,  // its length
  short *message_bit_lut,
  unsigned char *XOR_lut,
  long read_or_write
){

  // this is a crude first version... April 1996

  // we're goin with 16 control bits, and in this demo, we''l use all of them
  // to describe the raw message length as a short unsigned int

  //int *length_table = new int[15];
  //int *xblocks = new int[15];
  //int *yblocks = new int[15];
  int length_table[] = {16,24,32,48,64,96,128,192,256,384,512,768,1024,1536,3072};
  int xblocks[] =      { 8, 8, 4, 8, 4, 4,  2,  4,  2,  2,  1,  2,   1,   1,   1};
  int yblocks[] =      { 8, 8, 8, 8, 4, 8,  4,  4,  2,  4,  2,  2,   1,   2,   1};

  // find which length in the length table is next highest over current message_length
  long index=0;
  while( length_table[index] < message_length ){
      index++;
  }

  long xlength = (SIGNATURE_BLOCK_DIMENSION/2)/xblocks[index];  // length in bumps
  long ylength = (SIGNATURE_BLOCK_DIMENSION/2)/yblocks[index];
  long current_bit,kfoo,lfoo;
  long jump = SIGNATURE_BLOCK_DIMENSION;
  short actual_bit;
  long one;
  long i,j,k,l;
  short *pmessage_bit;
  unsigned char *pXOR;
  for(i=0;i<yblocks[index];i++){
      for(j=0;j<xblocks[index];j++){
          current_bit = 11*i + j*xlength;  // this is
              //simply a "mixing agent" so that given bit planes
              // don't congregate around edges (come up with a better way please please

          // the following uses the
          //       1  0
          //       0  1
          // formula of local bumps associated with a given bit plane, hence the 2's
          // floating around
          for(k=0;k<ylength;k++){
              // reset the pointers
              pmessage_bit = &message_bit_lut[2*j*xlength + 2*(i*ylength+k)*jump];
              pXOR = &XOR_lut[2*j*xlength + 2*(i*ylength+k)*jump];
              kfoo = (k+6)%8;
              for(l=0;l<xlength;l++){
                  lfoo = (l+6)%8;
                  if(kfoo<4 && lfoo <4){  // this is the 16 bit control code region
                      actual_bit = (short)(message_length + kfoo*4 + lfoo);
                  }
                  else {  // this is the embedded data region
                      actual_bit = (short)(current_bit % message_length);
                      current_bit++;
                  }
                  *pmessage_bit = *(pmessage_bit+1) = *(pmessage_bit+jump) =
                      *(pmessage_bit+jump+1) = actual_bit;
                  pmessage_bit+=2;

                  if(read_or_write)one = 1;
                  else{
                      if(actual_bit >= message_length){
                          if(control_message[actual_bit-message_length])one = 1;
                          else one = 0;
                      }
                      else {
                          if(message[actual_bit])one=1;
                          else one = 0;
                      }
                  }
                  if(one){
                      *pXOR = 1;
                      *(pXOR+1) = 0;
                      *(pXOR+jump) = 0;
                      *(pXOR+jump+1) = 1;
                  }
                  else {
                      *pXOR = 0;
                      *(pXOR+1) = 1;
                      *(pXOR+jump) = 1;
                      *(pXOR+jump+1) = 0;
                  }
                  pXOR+=2;
              }
          }
      }
  }

  //delete [] length_table;
  //delete [] xblocks;
  //delete [] yblocks;

  return(1);
}


int load_output_array(
    float *tweak,
    unsigned char *data_out,
    unsigned char *data,
    long xdim,
    long zdim,
    long bump_size,
    long jump_x
){
    unsigned char *pdata,*pdata_out;
    int i,j,k,temp;
    float *ptweak,half = (float)0.5;

    pdata = data;
    ptweak = tweak;
    pdata_out = data_out;
    if(zdim == 1){  // single channel
        if(bump_size == 1){
            for(j=0;j<xdim;j++){
                temp = (int)( (float)*(pdata++) + *(ptweak++) + half );
                if(temp<0)*(pdata_out++)=0;
                else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=HIGHEST_GREY_VALUE;
                else *(pdata_out++) = (unsigned char)temp;
            }
        }
        else {
            for(i=0;i<bump_size;i++){
                ptweak = tweak;
                for(j=0;j<xdim;j++){
                    for(k=0;k<bump_size;k++){
                        temp = (int)( (float)*(pdata++) + *ptweak + half );
                        if(temp<0)*(pdata_out++)=0;
                        else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=HIGHEST_GREY_VALUE;
                        else *(pdata_out++) = (unsigned char)temp;
                    }
                    ptweak++;
                }
```

```
            pdata += jump_x;
            pdata_out += jump_x;
        }
    }
}
else { // multi-channel, assume ONLY RGB and three channels at present
    float red = (float)RED_DOG,green=(float)GREEN_DOG,blue=(float)BLUE_DOG;
    float red_ratio,green_ratio,blue_ratio,lum,zero = (float)0.1;
    if(bump_size == 1){
        for(j=0;j<xdim;j++){
            lum = red * (float)*pdata + green * (float)*(pdata+1) + blue * (float)*(pdata+2);
            if(lum>zero){
                red_ratio = (float)*(pdata++) / lum;
                green_ratio = (float)*(pdata++) / lum;
                blue_ratio = (float)*(pdata++) / lum;
            }
            else {
                red_ratio = green_ratio = blue_ratio = (float)1.0;
                pdata+=3;
            }
            lum += *(ptweak++);
            // red
            temp = (int)( lum * red_ratio + half );
            if(temp<0)*(pdata_out++)=0;
            else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=(unsigned char)HIGHEST_GREY_VALUE;
            else *(pdata_out++) = (unsigned char)temp;
            // green
            temp = (int)( lum * green_ratio + half );
            if(temp<0)*(pdata_out++)=0;
            else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=(unsigned char)HIGHEST_GREY_VALUE;
            else *(pdata_out++) = (unsigned char)temp;
            // blue
            temp = (int)( lum * blue_ratio + half );
            if(temp<0)*(pdata_out++)=0;
            else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=(unsigned char)HIGHEST_GREY_VALUE;
            else *(pdata_out++) = (unsigned char)temp;
        }
    }
    else {
        for(i=0;i<bump_size;i++){
            ptweak = tweak;
            for(j=0;j<xdim;j++){
                for(k=0;k<bump_size;k++){
                    lum = red * (float)*pdata + green * (float)*(pdata+1) + blue *
(float)*(pdata+2);
                    if(lum>zero){
                        red_ratio = (float)*(pdata++) / lum;
                        green_ratio = (float)*(pdata++) / lum;
                        blue_ratio = (float)*(pdata++) / lum;
                    }
                    else {
                        red_ratio = green_ratio = blue_ratio = (float)1.0;
                        pdata+=3;
                    }
                    lum += *ptweak;
                    // red
                    temp = (int)( lum * red_ratio + half );
                    if(temp<0)*(pdata_out++)=0;
                    else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=HIGHEST_GREY_VALUE;
                    else *(pdata_out++) = (unsigned char)temp;
                    // green
                    temp = (int)( lum * green_ratio + half );
                    if(temp<0)*(pdata_out++)=0;
                    else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=HIGHEST_GREY_VALUE;
                    else *(pdata_out++) = (unsigned char)temp;
                    // blue
                    temp = (int)( lum * blue_ratio + half );
                    if(temp<0)*(pdata_out++)=0;
                    else if(temp>HIGHEST_GREY_VALUE)*(pdata_out++)=HIGHEST_GREY_VALUE;
                    else *(pdata_out++) = (unsigned char)temp;
                }
                ptweak++;
            }
            pdata += jump_x*zdim;
            pdata_out += jump_x*zdim;
        }
    }
  }
  return(1);
}


////////////////////////////////////////////////////////////////////////////
// core_sign_public_generation_1()
//
// problem has been reduced to basic block unit;
// the only special case is when xdim and/or ydim are not extended to full block size
////////////////////////////////////////////////////////////////////////////
int core_sign_public_generation1(
        unsigned char *data,   // pointer to upper left corner of image block
        long xdim,             // absolute pixel dimension of current block
        long Original_xdim,    // absolute pixel dimension of entire original image or passed array
        long ydim,             // absolute pixel dimension of current block
        long zdim,             // number of channels, e.g. 3 for RGB
        long bump_size,
        long message_length,  // message length
        short *message_bit_lut,
        unsigned char *XOR_lut, // this can be economized and reduced by 8 by using bitwise
packing (I don't bother here)
        float *luminance_lut,
        float *detail_lut,
        float *subliminal_grid,
        unsigned char *data_out,  // NULL if data is to be put back into input array
        float global_gain,
        float asymetric_gain,
        float *funky_lut
){
        long jump_x = Original_xdim - xdim;  // this is the pointer offset for jumping rows
        unsigned char *pdata_out;
        long i,j;
        float *p1,*p2,*p3,*p4,*pbump,local_average,gain,detail_gain,diff;
        float *psubliminal_grid,lum_gain,asym_gain,funky_gain;
        short *pbit;
        unsigned char *pXOR;
        double dtemp,bottomfunk;

        // set pdata_out based on (in place) versus new output array
        if(data_out == NULL)pdata_out = data;
        else pdata_out = data_out;

        // calculate bitwise bias between original image, (optionally degraded by common-model
        // distortion), and each bit of the message; this will be used for differential gain of
        // the bit planes to help "struggling" bits
        float *bit_bias = new float[message_length];
        for(i=0;i<message_length;i++)bit_bias[i]=(float)1.0;
        // read_block_signature(
        // convert_read_to_bias(

        // dive into main loop
        /*
        Main loop version 1 works in the following way.  It is designed so that it can
        create a lagged version of the output in order to support either case of: A) where
        the input data array is replaced with the output array (in place), or B) where the
        *data_out pointer is not null and is the actual output array.
        ___THIS PARTICULAR VERSION EXPECTS case B ____

        The main loop essentially operates bump by bump.  It determines the local overall
        gain that should be applied to the given bump, then tweaks the individual pixel(s)
        of the output bump and stores in the temporary array which is later written out into
        the ultimate output array.
        */
        long xbumpdim = xdim/bump_size;  // calling routine guaranteed this would never have a
remainder
        long ybumpdim = ydim/bump_size;
        // create initial bump arrays
        int xbumpsize = xbumpdim+2; // adding '2' allows us to not worry about edges in core loops
        float *bump0 = new float[xbumpsize];
        float *bump1 = new float[xbumpsize];
        float *bump2 = new float[xbumpsize];
        // load row 1 and row 2 (with row 0 data) for the first process step
        // load_bump_array should copy elements 0 and 1 with data bump 0
        // and elements xbumpdim and xbumpdim+1 with data bump xbumpdim-1
        load_bump_array(bump1,data,xbumpdim,zdim,bump_size,jump_x,1);
        memcpy(bump2,bump1,xbumpsize*sizeof(float));
        // create tweak array for each raster of bumps
        float *tweak = new float[xbumpdim];
        float *ptweak;
        float f1 = (float)1.0;
        float f4 = (float)4.0;

        for(i=0;i<ybumpdim;i++){
            // in order to avoid modulo housekeeping later on, copy the arrays downward
            //(as they are small too)
            memcpy(bump0,bump1,xbumpsize*sizeof(float));
            memcpy(bump1,bump2,xbumpsize*sizeof(float));
            if(i!=(ybumpdim-1)){ // load next bump row array

load_bump_array(bump2,&data[(i+1)*bump_size*Original_xdim*zdim],xbumpdim,zdim,bump_size,jump_x
,1);
            }
            else {} // leave bump2 alone

            p1 = bump0+1;
            p2 = bump1;
            p3 = bump2+1;
            p4 = bump1+2;
            pbump = bump1+1;
            psubliminal_grid = &subliminal_grid[i*SIGNATURE_BLOCK_DIMENSION];
            ptweak = tweak;
```

```
pbit = &message_bit_lut[i*SIGNATURE_BLOCK_DIMENSION];
pXOR = &XOR_lut[i*SIGNATURE_BLOCK_DIMENSION];
for(j=0;j<xbumpdim;j++){  // this is the heart of the signing code and process one bump at a
time

    /* Here's the deal: (Written 4/26/96)
    The goal of the signing process, beyond simply functioning,
    is to maximize the "numeric detectability" of an embedded signature
    while meeting some form of fixed "visibility/acceptability threshold" set by
    a given user/creator.

    In service to design toward this goal, imagine the following three axis
    parameter space, where two of the axes are only half-axes (positive only),
    and the third is a full axis (both negative and positive). This set of axes define
    two of the usual eigth octal spaces of euclidean 3-space. As things refine and
    "deservedly separable" paramters show up on the scene (such as "extended local
    visibility metrics"), then they can define their own (generally) half-axis and
    extend the following example beyond three dimensions.

    The signing design goal becomes optimally assigning a "gain" to a local
    bump based on its coordinates in the above defined space, whilst keeping in mind
    the basic needs of doing the operations fast in real applications. To begin
    with, the three axes are the following. We'll call the two half axes x and y,
    while the full axis will be z.

    The x axis represents the luminance of the singular bump. The basic idea is
    that you can squeeze a little more energy into bright regions as
    opposed to dim ones. It is important to note that when true "psycho-linear -
    device independent" luminance values (pixel DN's) come along, this axis might
    become superfluous, unless of course if the luminance value couples
    into the other operative axes (e.g. C*xy). For now, this is here as much due
    to the sub-optimality of current quasi-linear luminance coding.

    The y axis is the kitchen sink of "local hiding potential" of the
    neighborhood within which the bump finds itself. The basic idea is that
    flat regions have a low hiding potential since the eye can detect subtle changes
    in such regions, whereas complex textured regions have a high hiding potential.
    Long lines and long edges tend toward the lower hiding potential since "breaks
    and choppiness" in nice smooth long lines are also somewhat visible, while
    shorter lines and edges, and mosaics thereof, tend toward the higher hiding
    potential. These latter notions of long and short are directly connected
    to processing time issues, as well to issues of the engineering resources
    needed to carefully quantify such parameters. Developing the working model
    of the y-axis will inevitably entail one part theory to one part
    picky-artist-empiricism. As the parts of the hodge-podge y-axis become
    better known, they can splinter off into their own independent axes if
    its worth it.

    The z-axis is the "with or against the grain" axis which is the full axis - as
    opposed to the other two half-axes. The basic idea is that a given input bump
    has a pre-existing bias relative to whether one wishes to encode a '1' or a '0'
    at its location, which to some non-trivial extent is a function of the reading
    algorithms which will be employed, whose (bias) magnitude is semi-correlated
    to the "hiding potential" of the y-axis, and....,fortunately..., can be used
    advantageously as a variable in determining what magnitude of a tweak value is
    assigned to the bump in question. The concommitant basic idea is that when a bump
    is already your friend, or even your friend in a big way, then why mess with it
    much, whereas when it is your enemy or a big time enemy, then you want to squash
    it like a four year old discovering how flat slugs can get underfoot. The really
    cool thing here is that, in general, the latter squashing operation tends
    more toward a local blurring operation as opposed to a local sharpening
    operation, and thus has somewhat less visibility per numeric tweak unit.

    The above general description of the problem should suffice for many years.
    Clearly adding in chrominance issues will expand the defintions a bit,
    leading to a bit more signature bang for the visibility, and human
    visibility research which is applied to the problem of compression can
    equally be applied to this area but for diametrically opposed reasons.
    Facinating possibilities truly. But alas, I am required to crank out some
    pot-shot first system which needs must neglect vast areas of the above general
    arenas. Here are its principles.

    For speed's sake, local hiding potential will be calculated only
    based on a 3 by 3 neighborhood of pixels, the center one being signed and
    its eight neighbors. Beyond speed issues, there is also no data or coherent
    theory to support anything larger as well. The design issue boils down
    to canning the y-axis visibility thing, how to couple the luminance
    into this, and a little bit on the friend/enemy asymetry thing.
    My guding prinicles to start are simply to make a flat region
    zero, a classic pure maxima or minima region a "1.0" or the highest value,
    and to have "local lines", "smooth slopes", "saddle points" and whatnot
    fall out somewhere in between. In other words, let's pull out the darts
    and throw a few and see if any land on the board.

    The following code has six basic parameters that will be used:
    1) luminance
    2) difference from local average
    3) the asymetry factor (with or against the grain)
    4) minimum linear funkiness factor (our crude attempt at flat v. lines v. maxima)
    5) bit plane bias factor
    6) global gain (the user's single top level gain knob)
    Even this list above can get complicated in their inter-relations and
    especially in our current lack of expirimental data to support various
    specific formulas.

    1) luminance is straightforward
    2) difference from local average is also, and is rather important to
    our first generation stuff since it will directly eb involved in
    reading signatures (assuming we don't get fancy phase-only reading
    algorithms going).
    3) the asymetry factor is a single scalar applied to the "against the grain"
    side of the difference axis of number 2 directly above, as well and being
    modified by the minimum linear funkiness factor below. [Certainly
    it can eventually become a function of other variables if and
    when data and theory supports such].
    4) The minimum linear funkiness factor is admitedly crude but it should
    be of some service even in a 3 by 3 neighborhood setting. The idea is that
    true 2D local minima and maxima will be highly pertorbed along each of the four
    lines travelling through the center pixel of the 3 by 3 neighborhood, while
    a visual line or edge will tend to flatten out at least one of the four
    linear profiles. [The four linear profiles are each 3 pixels in length, i.e.,
    the top left pixel - center - bottom right; the top center - center - bottom
    center; the top right - center - bottom left; the right center - center - left
    center;]. Let's choose some metric of "funkiness" or entropy as applied to
    three pixels in a row, perform this on all four linear profiles, then choose
    the minimum value for our ultimate parameter to be used as our 'y-axis'. Cheers
    to she or he who will take all of this to the next levels of refinement.
    5) The bit plane bias factor is an interesting creature with two faces, the
    pre-emptive face and the post-emptive face. In the former, you simply "read"
    the unsigned image and see where all the biases fall out for all the bit planes,
    then simply boost the "global gain" of the bit planes which are, in total,
    going against your desired message, and leave the others alone or even slightly
    lower their gain. In the post-emptive modaliction, you churn out the whole
    signing process replete with the pre-emptive bit plane bias and the other 5
    parameters listed here, and then you e.g. run the signed image through heavy
    JPG compression AND model the "gestalt distortion" of line screen printing and
    subsequent scanning of the image, and...., then..., you read the image and find out
    which bit planes are struggling or even in error, you appropriately beef up
    the bit plane bias, and you run through the process again. If you have good
    data driving the beefing process you should only need to perform this step
    once, or, you can easily Van-Cittertize the process (arcane reference to
    reiterate the process with some damping factor applied to the tweaks).
    6) Finally, there is the global gain. the goal is to make this
    single variable be the top level "intensity knob" that the slightly curious
    user can adjust if they want to. The very curious user can navigate down
    advanced menus to get their epirimental hands on the other five variables here,
    and who knows what others in the future.

    whew, that's the most commenting I've ever done, I must be getting old or
    maybe I'm just realizing it would be nice to leave a signpost or two in
    this first dart throwing.
    */

    // get luminance gain
    lum_gain = luminance_lut[ (int)*pbump ];

    // find current differential between bump value and local average
    // this one can generally make use of inter-DN lut's:
    // in this case, down to 0.25 of a DN
    local_average = *p1 + *p2 + *p3 + *p4;
    diff = *pbump * f4 - local_average;
    detail_gain = detail_lut[ (int)( fabs( (double)diff ) ) ];

    // now calculate tweak based first on message, include asymetric gain
    if( *(pXOR++) ){
        if(diff<0.0)asym_gain = asymetric_gain;
        else asym_gain = f1;
        *ptweak = f1;   // slip this one in here
    }
    else {
        if(diff>0.0)asym_gain = asymetric_gain;
        else asym_gain = f1;
        *ptweak = -f1;
    }

    // funky time: minimum linear funkiness factor
    // line 1
    bottomfunk = fabs((double)(*pbump - *(p1-1))) + fabs((double)(*pbump - *(p3+1)));
    // line 2
    dtemp = fabs((double)(*pbump - *p1)) + fabs((double)(*pbump - *p3));
    if(dtemp <bottomfunk)bottomfunk = dtemp;
    // line 3
    dtemp = fabs((double)(*pbump - *(p1+1))) + fabs((double)(*pbump - *(p3-1)));
    if(dtemp <bottomfunk)bottomfunk = dtemp;
    // line 4
    dtemp = fabs((double)(*pbump - *p2)) + fabs((double)(*pbump - *p4));
    if(dtemp <bottomfunk)bottomfunk = dtemp;
    funky_gain = funky_lut[ (int)bottomfunk ];
```

```
            // add in the bias
            // *ptweak *= bit_bias[*(pbit++)];

            // now put them all together somehow, but how??
            gain = global_gain * (lum_gain + asym_gain * (funky_gain + detail_gain));
            *ptweak *= gain;

            // then add in subliminal grid
            // eventually make this subject to local gain as well
            if(gain > GRID_MINIMUM_GAIN) *ptweak += *psubliminal_grid;
            psubliminal_grid++;ptweak++;pbump++;p1++;p2++;p3++;p4++;
        }
        load_output_array(tweak,&pdata_out[i*bump_size*Original_xdim*zdim],
            &data[i*bump_size*Original_xdim*zdim],xdim,zdim,bump_size,jump_x);
    }

    // optionally JPEG compress (or whatever compress) the output buffer
    // find the new bit biases, fine tune the bit bias values and
    // repeat the above operations

    delete [] bit_bias;
    delete [] bump0;
    delete [] bump1;
    delete [] bump2;
    return(1);
}


/////////////////////////////////////////////////////////////////////////////
// sign_public_generation_1()
//
/////////////////////////////////////////////////////////////////////////////
int sign_public_generation_1(
    unsigned char *data,        // input data to be signed
  long xdim,                    // it's x dimension
  long ydim,                    // it's y dimension
  long zdim,                    // generally 1 for B&W and 3 for 3x8bit RGB, data assumed R-G-B
  long bump_size,               // number of pixels per singular bump along one dimension; e.g.2 for 2x2
bump size
  unsigned char *message,       // either 0 or 1, inefficient but simple
  long  message_length,         // length of message in BITS, also length of message string
  unsigned char *control_message,  // this is the separate "always gotta be there" message
  long control_message_length,  // its length
  float *luminance_lut,         // look up table mapping the scaling to luminance values
  float *detail_lut,            // look up table mapping the scaling to local detail
  float *subliminal_grid,       // this is the image of the subliminal grid, in the image domain
  unsigned char *data_out,      // signed output data in same length and format as input, NULL if output
is to be placed into input array 'data'
  float global_gain,
  float asymetric_gain
){
  long block_pixel_dimension,x_blocks,x_leftover,y_blocks,y_leftover,i,j,status=1;
  long temp,block_xdim,block_ydim;
  unsigned char *pdata,*pdata_out;

  block_pixel_dimension = SIGNATURE_BLOCK_DIMENSION * bump_size;  // actual pixel dimension of a
standard signature block
  x_blocks = 1+(xdim-1)/block_pixel_dimension;  // number of full (and possibly partial on the last)
basic blocks
  x_leftover = xdim%block_pixel_dimension - xdim%bump_size; // ignore fractional bumps on ends
  y_blocks = 1+(ydim-1)/block_pixel_dimension;
  y_leftover = ydim%block_pixel_dimension - ydim%bump_size; // ignore fractional bumps on ends
  // though the straggly bits on the ends can cause a bit of a bookkeeping issue, they save alot of
  // headaches when it comes time to write simple core algorithms sans if statements

  // load the message length into the 16 bit long control message
  int ii = 1;
  control_message_length = 16;
  for(i=0;i<16;i++){
      if(ii & (short)message_length)control_message[i] = 1;
      else control_message[i] = 0;
      ii *= 2;
  }

  //  BE SURE TO COPY END FRACTIONAL BUMP DATA FROM INPUT TO OUTPUT, UNCHANGED
  // in other words, if xdim%bump_size or ydim%bump_size is non-zero, then we can
  // immediately copy the leftmost and bottommost strip into the output buffer, unchanged
  if( data_out != NULL ){  // if data output buffer is the input buffer, no need for copying
      if( temp = (xdim%bump_size) ){
          for(i=0;i<ydim;i++){
              pdata = &data[zdim*((i+1)*xdim-temp)];
              pdata_out = &data_out[zdim*((i+1)*xdim-temp)];
              for(j=0;j<temp*zdim;j++)*(pdata_out++) = *(pdata++);
          }
      }
      if( temp = (ydim%bump_size) ){
          pdata = &data[(ydim-temp)*xdim*zdim];
          pdata_out = &data_out[(ydim-temp)*xdim*zdim];
          for(i=0;i<temp*xdim*zdim;i++)*(pdata_out++) = *(pdata++);
      }
  }
  // let's have a few more arrays
  int total = SIGNATURE_BLOCK_DIMENSION * SIGNATURE_BLOCK_DIMENSION;
  unsigned char *XOR_lut = new unsigned char[total];
  short *message_bit_lut = new short[total];
  float *funky_lut = new float[512];
  // In this first version, each message block will have the same mapping
  // of bump locations to message bit planes, as well as the associated XOR parameter
  load_standard_message_block_lut(message,message_length,control_message,
      control_message_length,message_bit_lut,XOR_lut,0); // last zero is for 'write'

  load_funky_lut(funky_lut);


  // chunk up image into basic blocks and call core signing routine
  for(i=0;i<y_blocks;i++){
      if( i==(y_blocks-1) && y_leftover)
      {
          block_ydim = y_leftover;
      }
      else block_ydim = block_pixel_dimension;
      for(j=0;j<x_blocks;j++){
          if( j==(x_blocks-1) && x_leftover)
          {
              block_xdim = x_leftover;
          }
          else block_xdim = block_pixel_dimension;

          // In the far distant future, when message protocols may not be precisely
          // repeated from one basic block to the next, then a function call will
          // be needed to load the specific message block look up tables

          // call core block processor with pointer to upper left hand corner of current
block
          // note: include original x dimension of data array for row stepping purposes
          core_sign_public_generation1(
              &data[(i*xdim+j)*block_pixel_dimension*zdim],
              block_xdim,
              xdim,
              block_ydim,
              zdim,
              bump_size,
              message_length,
              message_bit_lut,
              XOR_lut,
              luminance_lut,
              detail_lut,
              subliminal_grid,
              &data_out[(i*xdim+j)*block_pixel_dimension*zdim],
              global_gain,
              asymetric_gain,
              funky_lut
          );
      }
  }

  delete [] XOR_lut;
  delete [] message_bit_lut;
  delete [] funky_lut;

  return(status);
}
```